FACULTAD DE INFORMÁTICA

UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DE MÁSTER

MÁSTER UNIVERSITARIO EN SOFTWARE Y SISTEMAS

ESPECIFICACIÓN DEL COMPORTAMIENTO DE UN AVATAR EN UN MUNDO VIRTUAL MEDIANTE UNA ONTOLOGÍA

AUTOR: Roberto Carlos Guerrero Navarrete
TUTOR: Jaime Ramírez Rodríguez

JULIO, 2015

Resumen

El presente trabajo describe la construcción de una aplicación que controla a un *Non Player Character* (NPC), en un mundo virtual. La aplicación desarrollada, que tiene como nombre BotManager, realiza dos tareas fundamentales: 1) conectarse al repositorio de conocimiento, que en esta implementación es una ontología expresada en OWL, para obtener las acciones que debe realizar el NPC dentro del mundo virtual; y 2) ordenar al NPC que realice estas acciones en un mundo virtual creado con la plataforma OpenSimulator. BotManager puede tener variadas aplicaciones, por lo tanto puede ser usada como complemento en mundos virtuales aplicados a la educación, simulación, ocio, etc. Ahora bien, la principal razón que motivó el desarrollo del BotManager fue la de crear un sistema de demostración automática de tareas en un mundo virtual destinado a la educación/entrenamiento. De esta forma, un Sistema Inteligente de Tutoría integrado con un mundo virtual podría demostrar paso a paso a un estudiante cómo realizar una tarea en el mundo virtual.

La ontología que lee el BotManager extiende la ontología propuesta en la tesis "Una propuesta de modelado del estudiante basada en ontologías y diagnóstico pedagógico-cognitivo no monótono" de Julia Parraga en el 2011 (Ontología de Julia). La construcción y las pruebas del BotManager se llevaron a cabo en tres etapas: 1) creación de la Ontología de Acciones del NPC que extiende la Ontología de Julia; 2) diseño e implementación de la aplicación en C# que lee la ontología que contiene el plan de acción del NPC, y ordena al NPC realizar las acciones en el mundo virtual; y 3) pruebas de la aplicación con la práctica "preparación de una taza de cafe", que es parte de un Laboratorio Virtual de Biotecnología.

El BotManager se ha diseñado como una aplicación cliente que se conecta a un servidor de OpenSimulator. Por lo tanto, puede ejecutarse en una máquina distinta a la del servidor. Asimismo, en la implementación del BotManager se ha utilizado una librería gratuita denominada LibOpenMetaverse que permite controlar un NPC de forma remota.

Abstract

This paper describes the construction of an application that controls a *Non Player Character* (NPC), in a virtual world. The application developed, called BotManager, performs two main tasks: 1) the connection to the repository of knowledge, which in this implementation is an ontology expressed in OWL, and retrieving the actions to be performed by the NPC within the virtual world; and 2) commanding the NPC to perform these actions in a virtual world created with the OpenSimulator platform. BotManager can have diverse applications, therefore it can be used as a complement in virtual worlds applied to education, simulation, entertainment, etc. However, the main reason behind the development of BotManager was to create an automatic demonstration of tasks in a virtual world for education / training. Thus, a virtual world integrated with an Intelligent Tutoring Systems could demonstrate step by step to a student how to perform a task in the virtual world.

The ontology used by the BotManager extends ontology proposed in the thesis “A proposal for modeling ontologies based student and not monotonous teaching-cognitive diagnosis” by Julia Parraga in 2011 (Julia’s Ontology). Construction and testing of BotManager were conducted in three stages: 1) creation of the NPC Actions Ontology by extending the Julia’s Ontology; 2) design and implementation of the application in C# that reads the ontology containing the plan of action of the NPC, and commands the NPC to perform the read plan in the virtual world; and 3) testing of the application with the practice “preparing a cup of coffee”, which is part of a Virtual Laboratory of Biotechnology.

The BotManager has been designed as a client application that connects to an OpenSimulator server. Therefore, it can run on a different machine to the server. To implement the BotManager we have used a free library called libopenmetaverse that allows us to control a NPC remotely.

Agradecimientos

- Al Eterno,
- a mis padres,
- a mis hermanos,
- a mis amigos,
- a mi tutor.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

La forma clásica de compartir el conocimiento ha sido a través de la transferencia de información de persona a persona, o a través de los libros. Más adelante ya en la era digital, gracias al avance de la tecnología, tenemos acceso en Internet a cualquier tipo de conocimiento incluido recursos multimedia que facilitan el aprendizaje. La reciente aparición de los entornos virtuales orientados a la enseñanza permiten que la transferencia del conocimiento a las personas se convierta en un proceso más efectivo y económico. Esto es debido a que se puede recrear toda la infraestructura necesaria para el aprendizaje, de modo que el usuario puede virtualmente realizar una actividad a través de un avatar de una forma parecida a como si estuviera en el mundo real.

Es así como los entornos virtuales constituyen una herramienta educativa muy útil. Si un entorno virtual además se complementa con inteligencia artificial y *Humanos Virtuales Autónomos* (Luzardo and Hernández, 2010) entonces estaremos hablando de una herramienta educativa aún más valiosa. Esta combinación puede ser usada en ámbitos tales como la simulación, la enseñanza, el entrenamiento, el turismo, etc.

En este trabajo se construyó una aplicación llamada BotManager, que controla a un avatar o un *Non Player Character* (NPC) dentro de un mundo virtual. Las acciones que realiza el NPC se encuentran definidas en una ontología en OWL en forma de un plan de acciones. Así, el BotManager lee estas acciones de la ontología y manda a un NPC que las ejecute en el mundo virtual. El BotManager ha sido diseñado como una aplicación multipropósito, es decir, que puede ser usada no sólo para proporcionar ayuda a los estudiantes en un entorno virtual mediante demostraciones prácticas, sino también para mostrar una animación de un NPC dentro de un mundo virtual como parte de una película animada.

Las acciones que realiza el NPC se encuentran definidas en una ontología que extiende la ontología propuesta en la tesis “Una propuesta de modelado del estudiante basada en ontologías y diagnóstico pedagógico-cognitivo no monótono” (Ontología de Julia) (Párraga, 2011). El mundo virtual donde se desenvuelve el NPC está construido en OpenSimulator. El BotManager es una aplicación cliente desarrollada en C#, que se conecta a un servidor con OpenSimulator a través de la librería OpenLibMetaverse. La librería OpenLibMetaverse permiten controlar un avatar dentro de OpenSimulator tal como lo haría un humano a través de un cliente visualizador de OpenSimulator, como es el caso de Firestorm. El BotManager en ejecución realiza lo siguiente: 1) lee la ontología de

definición de acciones e instancias de estas acciones; y 2) ordena al NPC que ejecute las acciones en el orden especificado en el mundo virtual.

Finalmente, se decidió probar la aplicación con una práctica que requiriera todos los tipos de acciones que puede realizar el BotManager. Para ello, fue necesario: 1) especificar los datos de la ontología y acceso al servidor en el archivo de configuración global del BotManager; 2) ejecutar un servidor de OpenSimulator con el mundo virtual que implementa la práctica de la preparación de la taza de café; y 3) ejecutar un cliente visualizador del mundo virtual, como el Firestorm, para poder observar la acciones que realiza el NPC. En la ejecución, se observa al NPC dentro del mundo virtual, realizando paso a paso de forma autónoma la práctica completa de preparación de la taza de café.

A continuación, se explica brevemente el contenido de los siguientes capítulos de la memoria:

En el capítulo 2 se establece el estado de la cuestión dando una visión general de los entornos virtuales de aprendizaje y los mundos virtuales así como las principales aplicaciones de los NPCs, la plataforma OpenSimulator, el concepto de ontología, los lenguajes de ontologías RDF y OWL, las principales herramientas para trabajar con estos lenguajes y los Sistemas Inteligentes de Tutoría con especial énfasis en el Modelo del Estudiante.

En el capítulo 3 se hace el planteamiento del problema, se definen las acciones a modelar, se especifican las características que debe tener el plan de acción del NPC en una ontología y se especifica cómo debe ser el motor de ejecución de dicho plan.

En el capítulo 4 se detalla el desarrollo de la aplicación; está dividido en dos partes: el modelado de las acciones mediante una ontología y el diseño del motor de ejecución de las acciones.

En el capítulo 5 se exponen las pruebas realizadas con la aplicación desarrollada; está dividido en tres partes: listado de las acciones que tiene que realizar el NPC en el mundo virtual; creación de las instancias de la ontología; y ejecución del BotManager.

En los capítulos 6 y 7 se presentan las conclusiones de la construcción de la aplicación y se plantean posibles líneas de trabajo futuras.

Capítulo 2

Estado de la Cuestión

En esta sección revisaremos los conceptos de Entorno Virtual de aprendizaje y mundo virtual; las principales aplicaciones de los NPCs (educación/entrenamiento, juegos, etc.); la plataforma de mundos virtuales OpenSimulator (servidor y cliente); la librería openlibmetaverse; el concepto de ontología; los lenguajes de ontologías OWL/RDF; y algunas herramientas para trabajar con ontologías (protege, jena, etc.).

2.1. Entornos Virtuales de Aprendizaje y Mundos Virtuales

Un *Entorno Virtual de Aprendizaje (EVA)* según (Gros Salvat, 2004), es la creación de materiales informáticos de enseñanza y aprendizaje, que se basan en un sistema de comunicación a través de un ordenador. (Dillenbourg et al., 2002) argumenta que los EVAs, tienen las siguientes características fundamentales:

1. Un EVA es un un espacio diseñado con finalidades formativas.
2. Un EVA es un espacio social, interacciones educativas se producen en el entorno, convirtiendo así los espacios en lugares.
3. Un EVA está representado explícitamente.
4. Los alumnos participan activamente, además también son actores y colaboradores en la construcción del espacio virtual.
5. Un EVA pueden ser utilizado tanto en la educación a distancia como en la educación presencial.
6. Un EVA se compone de varias tecnologías y es pedagógicamente multi-propósito.
7. Un EVA se puede complementar con un entorno físico.

Una de las primeras definiciones de *Mundo Virtual* es la siguiente "Un Mundo es un entorno en el que sus habitantes se consideran autónomos. No tiene por qué ser un planeta entero: se utiliza en el mismo sentido como 'el Mundo Romano' o 'el mundo de las altas finanzas"' (Bartle, 2004). En (Riofrío Luzcando, 2012) se hace un breve análisis del origen de los *Mundos Virtuales*. Nacen de los *Multi-User Dungeons* (MUD) [1], el primero fue creado en 1978 por Roy Trusbshaw junto con Richard Bartle, en lenguaje ensamblador. El año 1961 aparece el sistema *Programmed Logic for Automatic Teaching Operations* (PLATO), utilizado en la enseñanza como tutores. Bartle menciona que en un mundo virtual existen ciertas reglas, que permiten a los participantes efectuar cambios en él. También menciona que, la interacción con el mundo virtual es mediante un avatar en tiempo real, así como, que es posible interactuar con más participantes mediante una red o Internet (Bartle, 2004).

A los *Mundos Virtuales* también se los conoce como *Metaversos*[2] (Criado and Tuset, 2014). Una definición más reciente de *Mundo Virtual*, se propone en (Messinger et al., 2008) y se define como "Una red síncrona, persistente de red de personas, representados como avatares(véase apartado 2.2), facilitado por equipos de red.

Algunos de los mundos virtuales más populares son:

1. Worl of Warcraft[3].
2. Second Life[4].
3. The Sims[5].
4. Habbo Hotel[6].
5. EverQuest[7].
6. Rune Scape[8].

2.2. Aplicaciones de los NPC

Un *avatar* es una representación digital (gráfica o textual), no se trata de una etiqueta o nombre solamente. Un avatar tiene la capacidad de realizar acciones, es controlado por un humano en tiempo real. Un personaje totalmente gráfico, creado en el videojuego *Age of Conan*(figura 2.1) es un avatar. Los avatares funcionan tal como una marioneta controlada por un usuario. Los usuarios envían comandos para controlar al avatar, pero es el avatar quien realiza la acción. Incluso las formas de

[1] Es otra forma de llamar a los Mundos Virtuales
[2] viene del inglés *metaverse*, la contracción de *meta universe*, es decir meta-universo
[3] http://eu.battle.net/wow/es
[4] http://secondlife.com/
[5] http://www.ea.com/es/sims
[6] https://www.habbo.com/
[7] https://www.everquest.com
[8] http://www.runescape.com/

Figura 2.1: Videojuego Age of Conan. Fuente: http://www.ageofconan.com/

comunicación, que vienen desde el usuario directamente, como chat de voz, son presentadas como acciones tomadas por el avatar (Messinger et al., 2008).

Por otro lado, un NPC (Non-Player Character (Personaje no jugador)), también llamado *Bot* o *Autómata*, es un personaje que no es controlado por una persona real. En un videojuego, este personaje es controlado por la computadora, a veces, mediante inteligencia artificial.

Los NPCs son usados en diferentes ámbitos tales como videojuegos, educación, simulación, etc.

2.2.1. Videojuegos

En los videojuegos (Johansson et al., 2014), los NPCs son los personajes que son "jugados" por el computador en lugar de un jugador humano. Los juegos proveen desafíos y experiencia inmersiva al jugador, donde los NPCs desempeñan un rol y tienen un comportamiento tan importantes como la impresión visual.

Un aspecto importante de los videojuegos es que sean desafiantes dando algún resultado que sea valorado por el jugador para que sea más interesante. Estos desafíos, también conocido como batallas, consisten en obstáculos asociados al cumplimiento de objetivos predefinidos en el juego. Los aspectos de las batallas están asociados con posibles recompensas con el fin de crear una experiencia interesante al jugador. Dentro de estas batallas, los NPCs son esenciales para crear una experiencia lo más creíble posible.

Otro ejemplo del uso de los NPCs y la utilización de inteligencia artificial, es el popular videojuego FIFA (figura 2.2), donde el usuario juega un partido de fútbol virtual. En este juego los usuarios controlan los avatares de un equipo, para competir con los NPCs que pertenecen al equipo contrario. En (Thurau et al., 2006) se clasificó por observación el comportamiento de los NPCs en cinco grupos:

a) Dribbling – un jugador tiene el balón y otro jugador lo esta siguiendo, b) Given-and-go – el balón va de ida y vuelta entre dos jugadores, c) *Ball retrieval* – un jugador corre hacia el balón en reposo e inesperadamente se lo lanza a otro jugador, d) *Long pass* – El balón fue enviado a una distancia muy larga y no regresó y e) *Solo* – el jugador está con el balón sin soltarlo, mientras es perseguido por otro jugador.

Figura 2.2: Videojuego FIFA 2015. Fuente: http://goo.gl/ioXkDd

2.2.2. Educación/Entrenamiento

Los NPCs pueden ser usados en la educación o entrenamiento. (Al-Gharaibeh and Jeffery, 2010) en su trabajo analizan el empleo de los NPCs, como tutores para misiones en juegos en línea dentro de mundos virtuales *(EVs)* multiusuarios. La figura 2.3 muestra un NPC en un Entorno Virtual Colaborativo (EVC), en una misión para enseñanza de Computación. Los NPCs son los "dadores de misiones" que presentan la línea de la historia a los usuarios en la aventura. Una bola roja sobre la cabeza en el EVC y un anillo rojo alrededor del brazo en Second Life [9], identifican a los tutores NPCs dentro del EVC con misiones como personas con quienes el usuario tiene una razón por la que interactuar con ellos.

[9]Second Life es un mundo virtual que consiste en un terreno dividido en regiones de tamaño fijo, donde los usuarios controlan avatares e interactúan entre ellos

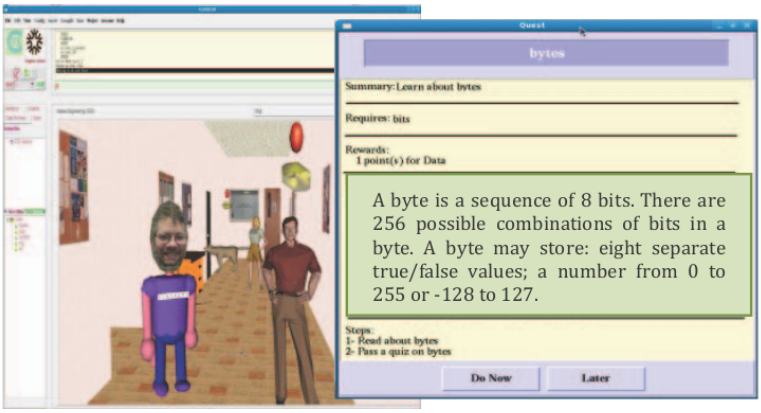

Figura 2.3: NPCs y usuarios en un EVC con ejemplo de una misión para enseñanza de Computación. Fuente: (Al-Gharaibeh and Jeffery, 2010)

Los NPCs son los "dadores de misiones" que presentan la línea de la historia a los usuarios en la aventura. La figura 2.3 muestra un NPC en un Entorno Virtual Colaborativo (EVC). Una bola roja sobre la cabeza en el EVC y un anillo rojo alrededor del brazo en Second Life [10], identifican a los tutores NPCs dentro del EVC con misiones como personas con quienes el usuario tiene una razón por la que interactuar con ellos.

Los tutores NPC usan dos tipos de conocimiento: lo que las misiones pueden ofrecer y que ellos recuerdan acerca de la experiencia de los usuarios en misiones pasadas y eventos del juego. La primera parte es casi estática y la segunda es dinámica y se acumula automáticamente durante el juego. El comportamiento del modelo de los tutores NPC es dirigido por un conjunto de reglas que determinan los movimientos de los NPCs ante estímulos externos. Existen cuatro tipos de comportamientos de NPC que son: estacionario (sin movimiento), rutinario (no especifica tareas con regularidad), vagabundo(se mueve aleatoriamente en un dominio dado) y compañero (acompaña a un jugador en un destino basado en la misión). Para especificar estos comportamientos es necesario un lenguaje de comportamiento, tal como ABL [11].

Otro uso de los NPCs es para entrenamiento en el campo de la odontología. En (Papadopoulos et al., 2013) crearon a "Erietta" un paciente virtual niño en un mundo virtual, aplicando gráficos en 3D, lenguaje LSL[12], principios de software educativos y gestión de la comunicación en odontología pediátrica. En la sala del tutorial (figura 2.4), se muestra una presentación interactiva en una pizarra, donde el usuario visualiza diapositivas acerca de técnicas básicas de comunicación y gestión de comportamiento en odontología pediátrica. Los modelos humanoide[13] de Erietta y su madre se posicionan junto a una unidad dental.

El escenario de aprendizaje, Erietta es una niña de ocho años de edad que junto a su madre visitan una clínica dental para la primera revisión oral por salud. La niña se sienta en la silla, para

[10] Second Life es un mundo virtual que consiste en un terreno dividido en regiones de tamaño fijo, donde los usuarios controlan avatares e interactúan entre ellos

[11] A Behavior Language (Un Lenguaje de Comportamiento)

[12] Lenguaje de Programación de Second Life

[13] Humanoide se refiere a cualquier ser cuya estructura corporal se asemeja a la de un humano

ser examinada por el dentista. El usuario enfrentará el miedo de la niña y la intervención de la madre. El objetivo es que la niña se relaje mediante el uso de técnicas básicas de comportamiento y comunicación. El escenario consta de seis partes a) bienvenida, b) interferencia, c)examen, d) distracción, e) rayos X y f)despedida. En cada parte se hacen preguntas de selección múltiple con sonido y texto de retroalimentación. Para cada respuesta correcta produce una reacción positiva de la niña, así como una respuesta negativa produce un comportamiento ansioso de la niña y se muestran recomendaciones para el usuario. Cuando se esta realizando correctamente los pasos, la niña se encuentra tranquila, se muestra el texto en color azul, mientras que cuando ocurre sentimientos negativos en la niña se muestra de color rojo.

Figura 2.4: Simulación de Erietta: Sala de tratamiento. Fuente: (Papadopoulos et al., 2013)

Los centros militares también están integrando los mundos virtuales para el entrenamiento de sus soldados, concretamente, la armada de los Estados Unidos (Korolov, 2013). La figura 2.5 muestra a un militar participando en un ejercicio de entrenamiento del proyecto MOSES, que simula las condiciones de un campo de batalla. MOSES tiene cerca de 3.000 acres cuadrados de tierra virtual, que pueden ser accedidos por cualquier usuario a través de Internet y un visor de mundos virtuales, tal como el *Firestorm*.

Figura 2.5: Militar de los Estados Unidos usando MOSES como entrenamiento militar. Fuente: (Korolov, 2013)

Una de las funciones principales de MOSES es facilitar la creación de prototipos de simulaciones virtuales, donde los entrenadores pueden, una vez autenticados en el sistema, diseñar entornos, probarlos con voluntarios y hacer cambios en tiempo real. Los NPCs en estas aplicaciones podrían ser camiones o tanques de guerra que siguen su trayectoria sin control humano, donde su comportamiento depende de las acciones que ocurren en el mundo virtual.

2.3. OpenSimulator

2.3.1. Servidor OpenSimulator

OpenSimulator[14] puede usarse para crear y alojar entornos o mundos virtuales 3D multiusuarios, los cuales pueden ser accedidos a través de una variedad de clientes, por medio del mismo protocolo, el protocolo de Second Life. De hecho, OpenSimulator puede ser usado para simular mundos virtuales similares a Second Life [15].

OpenSimulator está escrito en el lenguaje de programación C#, y puede ejecutarse en el sistema operativo Windows sobre la plataforma .Net o en los sistemas operativos linux o MacOs sobre el *framework* Mono. El código fuente está liberado bajo una licencia BSD [16]. La figura 2.6 muestra una pantalla desarrollada por la empresa Coolbleiben, que se dedica a realizar implementaciones de mundos virtuales en OpenSimulator.

OpenSimulator permite a los desarrolladores de mundos virtuales personalizar mundos usando su tecnología favorita. Además, el *framework* de OpenSimulator es fácilmente extensible.

[14] http://opensimulator.org/

[15] Es un meta-universo desarrollado por Linden Lab

[16] Berkeley Software Distribution, licencia de software libre permisiva

Figura 2.6: Pantalla de una implementación de OpenSimulator. Fuente: http://goo.gl/8bkUqw

OpenSimulator cuenta con las siguientes características:

- Implementa simulación física en tiempo real, con múltiples opciones de motores como el Bullet y ODE [17].
- Permite utilizar como lenguajes de *scripting* LSL/OSSL y C#.
- Los clientes pueden crear contenidos tridimensionales y programar scripts para los objetos 3D en tiempo real.
- Provee capacidad ilimitada para personalizar las aplicaciones del mundo virtual a través del uso de plugins.

Los objetos del mundo virtual que ejecutan scripts se pueden comunicar entre sí mediante el intercambio de mensajes.

OpenSimulator también cuenta con un módulo opcional el *Hipergrid* para permitir a los usuarios visitar otros servidores de OpenSimulator integrados en el mismo grid.

2.3.2. Clientes OpenSimulator

OpenSimulator permite utilizar diferentes visores o clientes para la visualización y creación de mundos virtuales. Uno de los más populares es Firestorm Viewer[18], que es un proyecto de la organización sin ánimo de lucro *The Phoenix Firestorm Project, Inc*. Firestorm Viewer ofrece una gran cantidad de funcionalidades para crear mundos virtuales para OpenSimulator. A través de un visor mundos virtuales, el usuario puede autenticarse al mundo virtual, controlar al avatar para realizar acciones tales de movimiento (caminar, correr o volar), coger o tocar objetos, creación de objetos, hablar con otros usuario a través del chat, etc.

2.4. LibOpenMetaverse

LibOpenMetaverse[19] es un conjunto de librerías de .NET escritas en lenguaje de programación C# para interactuar en entornos virtuales implementados en OpenSimulator.

LibOpenMetaverse implementa el protocolo de Second Life/OpenSimulator entre el cliente y el servidor, de manera que facilita la implementación de programas que manden órdenes o acciones a NPCs. Entre el conjunto de librerías de LibOpenMetaverse tenemos:

- *GridClient* para la creación de NPCs o clientes que que se conectan a un simulador de entornos virtuales (OpenSimulator).

[17] Son motores físicos de OpenSimulator
[18] http://www.firestormviewer.org/
[19] http://lib.openmetaverse.org/

- *TestClient* provee de una serie de ejemplos para operaciones comunes que un NPC requiera realizar.
- *WinGridProxy* provee un proxy para captura y decodificación de mensajes que fluyen entre el cliente y el simulador.
- *Simian* muestra una implementación simulador de ejemplo.
- Así como un conjunto de aplicaciones gráficas que muestran escenas de ejemplos renderizados[20].

2.5. Ingeniería Ontológica

Según (Gruber, 1993) una ontología es una especificación de una conceptualización. En el libro (Antoniou and Van Harmelet, 2004) una ontología describe formalmente un dominio del discurso. Normalmente, una ontología consiste en una lista finita de términos y relaciones entre estos términos. Los términos denotan *conceptos* importantes (*clases* de objetos) del dominio. Por ejemplo, en una universidad miembros del personal, estudiantes, cursos, aulas y disciplinas son algunos conceptos importantes.

La más típica *relación* entre clases es la de generalización. Aparte de las relaciones de generalización, las ontologías también pueden incluir información tal como:

- *Propiedades* – X enseña Y.
- *Restricciones de valor* – sólo los miembros de la facultad pueden enseñar cursos.
- *Declaraciones de clases disjuntas* – profesores y personal en general son disjuntos
- *Especificaciones de las relaciones lógicas entre los objetos* – todos los departamentos deben incluir al menos diez miembros de la facultad

En el contexto de la Web, una ontología provee una comprensión compartida de un dominio. Las ontologías pueden ser usadas en la organización y navegación de sitios Web. Como se aprecia en muchos sitios Web actuales, que en el lado izquierdo muestran los niveles superiores de una jerarquía de conceptos. El usuario puede expandir al dar clic en los conceptos para visualizar las subcategorías. También son muy usados para mejorar el rendimiento y precisión de los motores de búsquedas en Internet. Por ejemplo, si el motor no encuentra la información requerida puede sugerir una consulta más general.

Existen varios lenguajes de ontologías recomendados por la W3C, entre ellos se encuentran RDF y OWL. A continuación, vamos a presentar someramente estos dos lenguajes.

[20]Proceso de generar una imagen o vídeo mediante el cálculo de iluminación indirecta, partiendo de un modelo tridimensional

2.5.1. RDF

Conocido como un "lenguaje" RDF [21] es básicamente un modelo de datos*Data-Model*. El componente básico es una declaración o sentencia conocida como *tripleta* compuesta de tres partes que son sujeto, predicado y un objeto. RDF es independiente del dominio donde no hay suposiciones sobre un un dominio particular de uso.

2.5.2. Conceptos RDF

Los principales conceptos de RDF son recursos, propiedades y declaraciones.

2.5.2.1. Recursos

Los recursos pueden ser, autores, libros, editores, lugares, personas, criterios de búsqueda, etc. Cada recurso tiene una URI[22]. Esta URI puede ser una URL [23] o dirección, así como cualquier tipo de identificador único, que no necesariamente habilita el acceso a un recurso. Es decir, un esquema URI no es sólo una locación Web, también podría representar objetos como números de teléfono, posiciones geográficas, códigos ISBN, etc.

2.5.2.2. Propiedades

Las propiedades son un tipo especial de recursos, que describen relaciones entre recursos, como ejemplo: "escrito por", "edad", "título", etc. Estas propiedades también se representan por una URI. Se emplean URIs para identificar *cosas* y relaciones entre ellas. De esta forma se obtiene un único identificador, que reduce el problema de representación de datos homónimos.

2.5.2.3. Declaraciones

Las declaraciones afirman las propiedades de los recursos. Se conoce como una tripleta compuesta de un recurso, una propiedad y un valor. Los valores pueden ser recursos o átomos (cadena de caracteres).

2.5.3. Esquemas RDF

RDF es un lenguaje universal que permite describir recursos usando un vocabulario propio. RDF no asume nada acerca de algún dominio de aplicación específico. Donde se realiza este acto es el

[21] Resource Description Framework
[22] Uniform Resource Identifier
[23] Uniform Resource Locator

Esquema RDF[24] *(RDFS)*. RDFS es similar a un esquema XML [25], pero no es exactamente lo mismo. XML especifica limitaciones en la estructura de documentos XML, en cambio los esquemas RDF definen el vocabulario que puede ser usado en los modelos de datos de RDF. En RDFS se definen vocabularios, especificando qué propiedades aplicar a un tipo de objetos, qué valores pueden tomar dichas propiedades y describiendo las relaciones entre objetos. Por ejemplo, se puede escribir:

> *Profesor* es una subclase de *miembro del personal académico*

Esta declaración define que un profesor es también un miembro de personal académico, la que deberá ser interpretado así por cualquier software de procesamiento RDF. Con el uso de semántica RFD/RDFS se pueden modelar dominios específicos. A continuación un ejemplo de esquema RDF con los siguientes elementos XML:

```
<miembroPersonalAcademico>Luis Palau</miembroPersonalAcademico>
<profesor>Dante Caceres</profesor>
<clase nombre="Matematicas Discretas">
  <esEnseadoPor>David Thomas</esEnseadoPor>
</clase>
```

Si se desea obtener todos los miembros de personal académico, la expresión con el lenguaje de construcción de expresiones de procesamiento de documentos XML, Xpath[26] sería:

> //miembroPersonalAcademico

Retornaría el resultado único Luis Palau. Desde la perspectiva de XML esta respuesta no es semánticamente correcta, ya que "Dante Caceres" y "David Thomas", deberían estar en la respuesta debido a que: todos los profesores son miembros del personal académico, es decir un *profesor* es una subclase de *miembroPersonalAcademico* y las clases son sólo enseñadas o impartidas por miembros del personal académico. Esta información usa un modelo semántico de un dominio especifico (personal de la universidad), que no se puede representar en XML o RDF, pero sí en RDFS, ya que es la forma como se representa el conocimiento escrito en un esquema RDF.

2.5.3.1. Clases Principales

Las clases principales de RDF son:

- *rdfs:Resource* – es la clase de todos los recursos.
- *rdfs:Class* – es la clase de todas las clases.
- *rdfs:Literal* – es la clase de todos los literales (cadena de texto).

[24] RDF Schema
[25] eXtensible Markup Language
[26] XML Path Language

- *rdfs:Property* – es la clase de todas las propiedades.
- *rdfs:Statement* – es la clase de todas las declaraciones.

El siguiente ejemplo muestra la clase *conferenciante*:

```
<rdfs:Class rdf:ID="conferenciante">
...
</rdfs:Class>
```

2.5.3.2. Propiedades Principales

Las propiedades principales de RDF para definir relaciones son:

- *rdf:type* – relaciona un recurso a la clase.
- *rdf:subClassOf* – relaciona una clase a una de sus superclases. Las instancias de la clase son instancias de su superclase.
- *rdf:subPropertyOf* – relaciona una propiedad a una de sus superpropiedades.

Un ejemplo declarando que todos los conferenciantes son miembros del personal académico:

```
<rdfs:Class rdf:about="conferenciante">
  <rdfs:subClassOf rdf:resource="miembroPersonalAcademico"/>
</rdfs:Class>
```

2.5.3.3. Propiedades Principales para definir Restricciones de Propiedades

Las principales propiedades para definir restricciones sobre propiedades son:

- *rdfs:domain* – establece el dominio de una propiedad textitP, el cual es la clase de los recursos que pueden mostrarse como sujetos en una tripleta con predicado textitP. Si no se especifica el dominio, ningún recurso es el sujeto.
- *rdfs:range* – especifica el rango de una propiedad textitP, el cual es la clase de los recursos que pueden mostrarse como valores en una tripleta con predicado textitP.

En el siguiente ejemplo, se declara que *telefono* tiene como dominio a los miembros del personal académico únicamente y como rango un literal RDF:

```
<rdf:Property rdf:ID="telefono">
  <rdfs:domain rdf:resource="#miembroPersonal"/>
  <rdfs:range rdf:resource="&rdf;Literal"/>
</rdf:Property>
```

2.5.4. OWL

El Lenguaje de Ontología Web o *OWL* surge como respuesta a las limitaciones que tienen RDF y RDFS, al definir predicados binarios, una jerarquía de subclases y una jerarquía de propiedades con dominio y definición de rango de estas propiedades. OWL safisface mejor que RDF los principales requisitos que debe reunir un lenguaje de ontologías: una sintaxis bien definida; soporte para realizar inferencia de forma eficiente; una semántica formal; suficiente potencia expresiva; y conveniencia de expresión.

2.5.4.1. Sublenguajes de OWL

OWL cuenta con tres diferentes sublenguajes que son:

Full OWL La totalidad del lenguaje es llamado Full OWL y usa todas las primitivas del lenguaje, así como la combinación libre de estas con RDF y RDFS. Un ejemplo sería en Full OWL, aplicar una restricción de cardinalidad de la clase de todas las clases que se pueden describir en toda la ontología. OWL es completamente compatible hacia arriba con RDF, sintáctica y semánticamente, lo que significa que cualquier documento RDF válido lo es también en un documento Full OWL. Así como cualquier conclusión RDF/RDFS es una conclusión OWL válida.

DL OWL o *Lógica Descriptiva*[27] es un sublenguaje de Full OWL que restringe la forma como los constructores de OWL y RDF pueden ser usados. Consiste básicamente en deshabilitar la aplicación del constructor OWL para que se corresponda con una descripción lógica bien estudiada, permitiendo de este modo el soporte para realizar razonamiento eficiente. Por otro lado, se pierde la total compatibilidad con RDF. Un documento RDF será extendido de algunas formas y restringido en otras antes de ser un documento DL OWL válido, así como cada documento DL OWL será un válido documento RDF.

Lite OWL Restringe aún más el DL OWL a un subconjunto de constructores del lenguaje. OWL excluyendo las clases enumeradas, declaraciones de clases disjuntas y cardinalidad arbitraria. Quedando de esta manera un lenguaje más simple, fácil de entender y aplicar para los usuarios, pero menos expresivo.

OWL mantiene el uso de RDF y RDFS en gran medida:

- Los tres tipos de OWL usan la sintaxis de RDF.
- Las instancias son declaradas como en RDF, usando descripciones RDF y añadiendo información.
- Constructores de OWL como *owl:Class*, *owl:DatatypeProperty* y *owl:ObjectProperty* son especializaciones de sus homólogos RDF.

[27] Description Logic

2.5.4.2. El lenguaje OWL

Sintaxis
Basado en RDF y RDFS, OWL utiliza la sintaxis basada en XML de RDF. Ya que RDF/XML no ofrece una sintaxis de fácil lectura, OWL define:

- Una sintaxis basada en XML[28] que no sigue la convención de RDF y es más legible para las personas.
- Un sintaxis abstracta que se utiliza en el *Documento de Especificación del Lenguaje* [29] mucho más reducida y legible que la sintaxis XML o RDF/XML.
- Sintaxis gráfica basada en UML[30].

Encabezado
Los documentos OWL o *Ontologías OWL* son documentos RDF. El elemento raíz de una ontología OWL es un elemento *rdf:RDF*, el cual especifica un número de *namespaces* o espacios de nombres:

```
<rdf:RDF
  xmlns:owl ="http://www.w3.org/2002/07/owl#"
  xmlns:rdf ="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:xsd ="http://www.w3.org/2001/XMLSchema#">
```

Una ontología OWL podría empezar con un conjunto de aserciones que se agrupan en un elemento *owl:Ontology* el cual contiene comentarios, control de versiones y la inclusión de otras ontologías. Como en el caso:

```
<owl:Ontology rdf:about="">
  <rdfs:comment>Ejemplo de una ontologa</rdfs:comment>
  <owl:priorVersion rdf:resource="http://http://www.fi.upm.es/TFM/ontologia
  <owl:imports rdf:resource="www.fi.upm.es/TFM/personas"/>
<rdfs:label>Ontologa Universidad</rdfs:label>
</owl:Ontology>
```

Elementos Class
Los elementos clase se definen usando el elemento *owl:Class*, como se ve a continuación:

```
<owl:Class rdf:ID="profesorAsociado">
  <rdfs:subClassOf rdf:resource="#miembroPersonalAcademico"/>
</owl:Class>
```

Para decir que la clase es disjunta de *profesorAsistente* se utiliza el elemento *owl:disjointWith*.

[28] Definido por la World Wide Web Consortium (W3C)
[29] Definido por la World Wide Web Consortium (W3C)
[30] Unified Modeling Language

```
<owl:Class rdf:about="#profesorAsociado">
  <owl:disjointWith rdf:resource="#profesorAsistente"/>
</owl:Class>
```

La equivalencia de clases es mediante el elemento *owl:equivalentClass*

```
<owl:Class rdf:ID="facultad">
  <owl:equivalentClass rdf:resource="#miembroPersonalAcademico"/>
</owl:Class>
```

Existen dos clases predefinidas que son *owl:Thing* y *owl:Nothing*, la primera contiene todo y la segunda es una clase vacía. Por consiguiente cualquier clase es subclase de *owl:Thing* y superclase de *owl:Nothing*.

Elementos Property
OWL tiene dos tipos de propiedades:

- Propiedades de objeto o *Object Properties* – que relaciona los objetos con otros objetos. Como por ejemplo *esEnseñadoPor* y *supervisa*.
- Propiedades de tipo de datos o *Data Type Properties* – relaciona los objetos con los valores de tipos de datos, como por ejemplo: *telefono*, *titulo* o *edad*.

Ejemplo de propiedades de tipos de datos:

```
<owl:DatatypeProperty rdf:ID="edad">
  <rdfs:domain rdf:resource="#persona"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#nonNegativeInt
</owl:DatatypeProperty>
```

Los tipos de datos pueden agruparse en un esquema XML y ser usados en una ontología OWL.

Ejemplo de una propiedad de objeto:

```
<owl:ObjectProperty rdf:ID="esEnseadoPor">
  <rdfs:domain rdf:resource="#clase"/>
  <rdfs:range rdf:resource="#miembroPersonalAcademico"/>
  <rdfs:subPropertyOf rdf:resource="#implica"/>
</owl:ObjectProperty>
```

Existe la "propiedad inversa" como en el caso de *esEnseñadoPor* y *enseña*:

```
<owl:ObjectProperty rdf:ID="ensea">
  <rdfs:range rdf:resource="#curso"/>
  <rdfs:domain rdf:resource="#miembroPersonalAcademico"/>
  <owl:inverseOf rdf:resource="#esEnseadoPor"/>
</owl:ObjectProperty>
```

Restricciones de Propiedades
Si se necesita especificar que la clase *C* cumple ciertas condiciones, lo cual significa que todas las instancias de *C* cumplen estas condiciones, equivale a decir que *C* es una subclase de *C'*, y que *C'* agrupa todos los objetos que satisfacen estas condiciones. La declaración de restricciones posibles son *owl:allValuesFrom*, *owl:hasValue* y *owl:someValuesFrom*. Por ejemplo, pueden declararse que los cursos de matemáticas son enseñados por Leonardo Arce, como se muestra a continuación:

```
<owl:Class rdf:about="#cursoMatematicas">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#esEnseadoPor"/>
      <owl:hasValue rdf:resource="#240062"/>
    </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>
```

Donde *owl:hasValue* define un valor específico que la propiedad especificada por *owl:onProperty* debe tener.

Para declarar, que todos los miembros del personal académico deben enseñar al menos un curso de ingeniería, se puede especificar así:

```
<owl:Class rdf:about="miembroPersonalAcademico">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#ensea"/>
      <owl:someValuesFrom rdf:resource="cursoIngenieria"/>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

Otro tipo de restricción es la cardinalidad, como se muestra a continuación:

```
<owl:Class rdf:about="curso">
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#esEnseadoPor"/>
      <owl:minCardinality rdf:datatype="&xsd;nonNegativeInteger">1</owl:minC
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
```

En este ejemplo se define el valor de "1" la cardinalidad y sera tratado como un entero no negativo, por el namespace *xsd*.

Propiedades Especiales
Al definir una propiedad se puede declarar como perteneciente a uno o más tipos de propiedades:

- *owl:TransitiveProperty* – define una propiedad transitiva, como “tiene mejor curso que”, “es más alto que’ o ”es un ancestro de“.
- *owl:SymetricProperty* – define una propiedad simétrica, tal como ”tiene el mismo curso de“ o ”es hermano de“.
- *owl:FunctionalProperty* – define una propiedad que puede contener máximo un valor para cada objeto, como ”edad“, ”ancho“, ”jefeDirecto“.
- *owl:InverseFunctionalProperty* – define un propiedad para dos objetos diferentes que no pueden tener el mismo valor. Como la propiedad ”esElNumeroDeDNIPara“, el número de DNI de una persona es único.

Un ejemplo de estas propiedades puede ser:

```
<owl:ObjectProperty rdf:ID="tieneElMismoCursoQue">
  <rdf:type rdf:resource="&owl;TransitiveProperty" />
  <rdf:type rdf:resource="&owl;SymmetricProperty" />
  <rdfs:domain rdf:resource="#estudiante" />
  <rdfs:range rdf:resource="#estudiante" />
</owl:ObjectProperty>
```

Instancias
Las instancias son declaradas como en RDF:

```
<rdf:Description rdf:ID="233463">
  <rdf:type rdf:resource="miembroPersonalAcademico"/>
</rdf:Description>
```

Añadiendo más detalles se tiene:

```
<miembroPersonalAcademico rdf:ID="233463">
  <uni:edad rdf:datatype="&xsd;integer">27</uni:edad>
</miembroPersonalAcademico>
```

2.5.5. Herramientas

2.5.5.1. Jena

Jena[31] es una API[32] para el lenguaje Java, que funciona en plataforma .Net a través del proyecto *Jena .Net Framework* de *Linked Data Tools*. Contiene una serie de tareas comunes para manejo de Ontologías. Jena puede ejecutarse dentro de la plataforma .Net o a través de línea de comandos. Jena puede trabajar con los lenguajes de ontologías OWL o RDFS y trae incluido un conjunto de ejemplos al respecto.

A continuación un ejemplo básico de creación de Ontología con Jena:

[31] http://jena.apache.org/
[32] Application Programming Interface

```
OntModel m = ModelFactory.createOntologyModel();
OntDocumentManager dm = m.getDocumentManager();
dm.addAltEntry( "http://www.eswc2006.org/technologies/ontology",
            "file:" + JENA + "src/examples/resources/eswc-2006-09-21.rdf"
m.read( "http://www.eswc2006.org/technologies/ontology" );
```

Entre las funciones principales que ofrece Jena para trabajar con archivos OWL/RDF están:

- Crear una ontología.
- Escribir datos en la ontología.
- Cargar una ontología existente.
- Listar todas las clases de la ontología.
- Listar las subclases de una clase.
- Obtener las propiedades de una clase.
- Listar las instancias de una clase y sus propiedades.

Las consultas de la ontología pueden realizarse a través de objetos de Jena o con el uso del lenguaje de consultas *SPARQL*[33]

2.5.5.2. Top Braid Composer

Top Braid Composer[34] es un IDE para manejo de ontologías basado en Eclipse[35]. Top Braid Composer se utiliza para desarrollar ontologías, configuraciones de integración de *data sources* [36] y creación de servicios Web e interfaces de usuario. Entre las funcionalidades que posee se encuentran las siguientes:

- Edición de ontologías y datos RDF.
- Desarrollo de Aplicaciones de Web Semántica.
- Herramientas de desarrollo de aplicaciones.
- Integración con TopBraid Live, que es un servidor de aplicaciones de negocio inteligente.
- Importación de fuentes de datos.
- Exportación de fuentes de datos.
- Soporte para XML y archivos de tablas.

[33] SPARQL Protocol and RDF Query Language es un lenguaje estandarizado para la consulta de grafos RDF
[34] http://www.topquadrant.com/
[35] IDE para desarrollo de aplicaciones Java
[36] Fuente o Origen de Datos

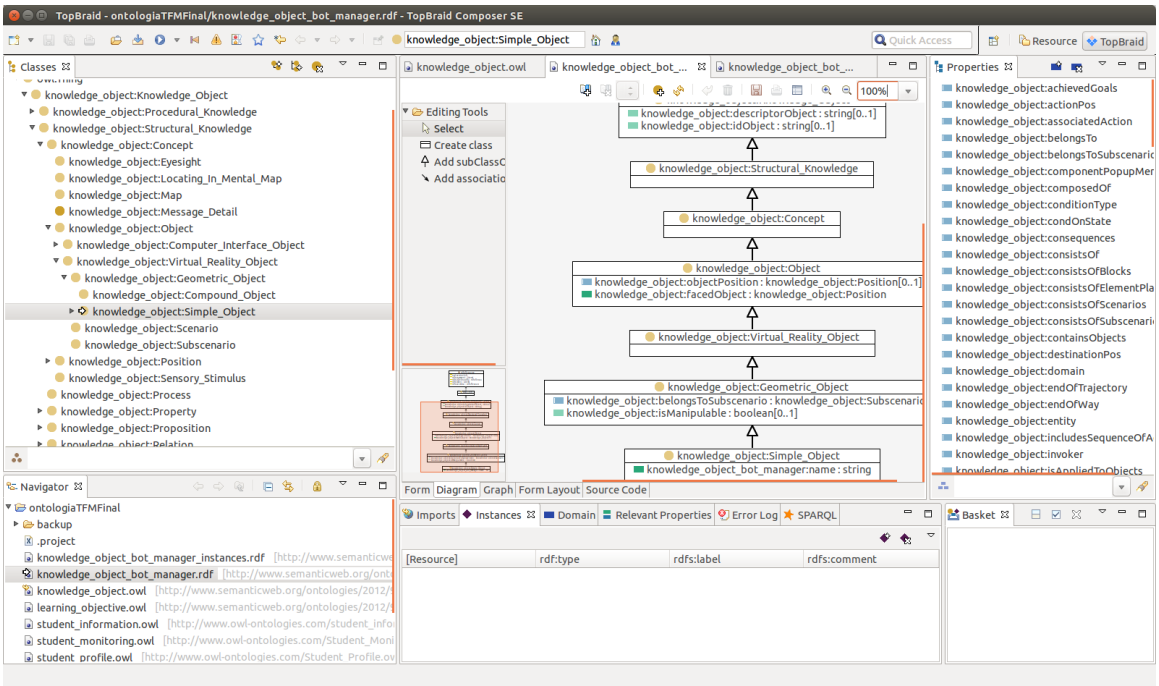

Figura 2.7: Pantalla Principal de Top Braid Composer editando una ontología

- Desarrollo de Datos Web.
- Integración de Datos.

La figura 2.7 muestra la edición de una ontología en Top Braid Composer.

2.6. Sistemas Inteligentes de Tutoría

Según (Párraga, 2011), una definición acertada de Sistema Inteligente de Tutoría (SIT) sería: "Un SIT es un sistema de software que, tratando de imitar la función de un tutor humano, y adaptándose al comportamiento del estudiante, tiene como objetivo orientar y facilitar el proceso de aprendizaje, ofreciéndole la ayuda que requiera en todo momento". Esta definición conlleva que un SIT tiene las siguientes características:

- El SIT se adapta al sistema de aprendizaje personalizado o a medida.
- Existe mejor retroalimentación con el estudiante, pudiendo esta ser a través de preguntas y respuestas.
- El sistema está alerta de los posibles errores que cometa el estudiante y de esta forma refuerza los aspectos en donde el estudiante requiera más entrenamiento.

De esta forma un SIT tiene la capacidad de decidir cuándo proveer al estudiante ayuda o enseñarle como realizar una tarea. Un SIT puede asumir el rol de un sistema experto [37] que puede contestar las dudas que tenga un estudiante y hacer un seguimiento en la realización de una tarea. Es importante que un SIT cuente con una interfaz amigable y evite también que el estudiante se distraiga haciendo que el aprendizaje sea entretenido.

2.6.1. Arquitectura de un Sistema Inteligente de Tutoría

Los SIT están compuestos por cuatro módulos que son:

- *Módulo Experto* – Es donde se encuentra el conocimiento de la materia a enseñar, resuelve las dudas como un profesional partiendo de una base de conocimientos de lo que el estudiante va a aprender.
- *Módulo del Estudiante* – Abarca información relacionada con el estudiante y la forma como este aprende un área de conocimiento. El SIT busca la mejor forma de interactuar con el estudiante para que este reciba conocimiento de acuerdo a su progreso, midiendo internamente su evolución. Esto se conoce como *Diagnóstico Cognitivo*.
- *Módulo de Tutoría* – Es el módulo que interpreta al maestro guía o tutor, debe poder aplicar una estrategia de aprendizaje a un ritmo adecuado, tal como lo haría un tutor humano. Por consiguiente encapsula la lógica necesaria para la interacción con el estudiante y debe tener la habilidad de decidir qué material se le muestra al estudiante, qué cosas sugerir o preguntar y cuándo indicarle que está cometiendo un error o se está desviando de su objetivo.
- *Módulo de comunicación* – Es el módulo de interacción entre el estudiante y el sistema. Implementa la interfaz con el estudiante, y entre otras cosas, se encarga de depurar las respuestas para evitar duplicidades y de facilitar el funcionamiento de los otros tres módulos.

2.6.2. Clasificación de los Sistemas Inteligentes de Tutoría

Los SITs pueden clasificarse de las siguientes formas:

- *Sistemas de Diagnóstico* – Indican al estudiante los errores cometidos en la resolución de un problema en el proceso de aprendizaje. Poseen una interfaz, un subsistema de resolución de problemas específicos, un sistema de diagnóstico de errores y un modelo del estudiante. Se los conoce también como *Sistemas Inteligentes de Ejercitación*.
- *Sistemas de Control* – Son usados para dirigir la capacidad cognitiva del estudiante. Son una extensión de los SIT de diagnóstico, abarcan el conocimiento específico junto con el alcance del aprendizaje esperado y técnicas de enseñanza (de conceptos y habilidades).

[37] Un sistema que emula el razonamiento de un experto en un dominio concreto

- *Sistemas de Supervisión* – Hacen un seguimiento de lo que realiza el estudiante y proporcionan ayuda cuando el estudiante falla o realiza un procedimiento inesperado. Son sistemas que evalúan lo que el estudiante realiza para establecer si necesita ser guiado o corregido. Este tipo de SIT es conocido como una extensión de los SITs de diagnóstico.

2.7. Modelo del Estudiante

En (Moreno Sabido, 2002) describen al Modelo del Estudiante como "una representación abstracta del estudiante". Es por ende un conjunto de ideas entre el profesor y el estudiante. En los Sistemas computacionales, el Modelo del Estudiante es el retrato abstracto del estudiante en la aplicación. De forma más concreta, un Modelo del Estudiante es el conocimiento del sistema respecto del estudiante, o dicho de otro modo, lo que el sistema cree que el estudiante cree. Se puede razonar sobre las creencias del sistema respecto a lo que conoce el estudiante de distintas formas. A partir de un historial del comportamiento del estudiante como origen de datos, se pueden hacer inferencias respecto a su conocimiento.

Existen una serie de aspectos que son la base para el modelado del estudiante tales como:

- *Deseos* – Lo que satisface al usuario personalmente.
- *Intenciones* – En base a las metas que quiera alcanzar el estudiante.
- *Capacidades* – El sistema considera las habilidades y experiencia del estudiante.
- *Intereses* – Las cosas que el estudiante realiza en la aplicación.
- *Fase de desarrollo del ego del estudiante* – Proveer recompensa o estímulo cuando el estudiante realiza bien una tarea.

2.7.1. Tipos de Modelos del Estudiante

Existen dos tipos de Modelo del Estudiante *explícitos* e *implícitos*.

- *Explícito* – Es una imagen del estudiante dentro del sistema de entrenamiento o aprendizaje, que sirve al sistema para decidir como se va a guiar al estudiante en el proceso de aprendizaje. Este modelo de sistemas son ideales para interactuar con el estudiante de forma personalizada. El sistema almacena el conocimiento respecto del estudiante y lo utiliza para adaptarlo a su necesidad.
- *Implícito* – Son directrices de diseño que surgen en base al análisis del estudiante por parte de quienes crean el sistema. Este modelo toma las decisiones en base a la observación del comportamiento del estudiante.

2.7.2. Construcción de un Modelo del Estudiante

En (González and Duque, 2008) plantean un modelo del estudiante, con ocho puntos que deben definirse para la creación de un modelo del estudiante en Sistemas Adaptativos de Educación Virtual:

- *Datos personales* – Es la información del estudiante que lo identifica dentro del sistema, como son el número de identificación, nombre, fecha y lugar de nacimiento, sexo, dirección, etc., que permiten al sistema identificar las preferencias del usuario, tales como el idioma o la apariencia del entorno que se le presentará al usuario.
- *Estado Anímico* – Referencia el estado actual del estudiante dentro del proceso de aprendizaje y los encasillan en uno de los siguientes tres estados: 1)*Positivo*: cuando el estudiante está confiado y motivado; 2)*Neutral*: el usuario es indiferente no muestra acciones positivas o negativas o 3)*Negativo*: el estudiante no coordina correctamente, tiene estrés o se encuentra irritable.
- *Contexto* – Trata acerca de la tecnología, como por ejemplo, la velocidad de la red, que delimitan el tipo de información multimedia que se le puede presentar al usuario.
- *Ambiental* – Corresponde a la interfaz del sistema, como tamaño o tipo de letras, temperatura o estado del tiempo que afectan al estado anímico del estudiante.
- *Estilos de Aprendizaje* – Es la forma como el estudiante aprende, denotado en sus habilidades, intereses y destrezas académicas.
- *Personalidad* – Es la característica que define como actúa el estudiante, usado para decidir cuándo este debe ser evaluado.
- *Académico* – El grado de conocimiento del tema que está aprendiendo, historial de visitas a recursos, tiempo que emplea en revisar el material, y registro de las acciones que realiza en el aprendizaje.
- *Psicológicos* – Define el tipo de inteligencia del estudiante, entre otros fines, para seleccionar el canal más adecuado por el que debe recibir la información.

Capítulo 3

Planteamiento del Problema

3.1. Introducción

Se requiere crear una aplicación multipropósito para controlar un NPC dentro de un mundo virtual. La actividad que realizará este NPC consiste en un plan o secuencia de acciones que estarán definidas en una ontología, al estilo de como si se tratase de un guión cinematográfico[1]. Para conseguir este objetivo es necesario realizar las siguientes tareas (ver figura 3.1):

1. Implementar una ontología que permita definir las acciones a realizar por un NPC en un mundo virtual creado con la plataforma OpenSimulator, y que extienda la ontología del estudiante de Julia (Párraga, 2011).
2. Realizar una aplicación en un lenguaje de programación de alto nivel que lea la ontología que contiene las acciones, se conecte a un mundo virtual creado con la plataforma OpenSimulator y ordene a un NPC que ejecute estas acciones.
3. Probar la aplicación con un plan y un mundo virtual concretos que resulten lo suficientemente representativos de los distintos tipos de acciones que se van a considerar.

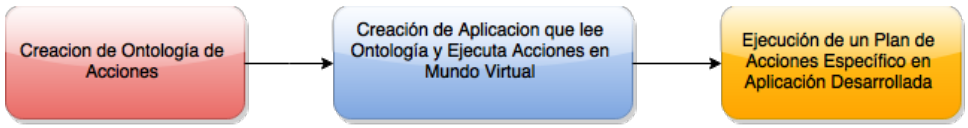

Figura 3.1: Acciones a Modelar

[1] Es el relato escrito de lo que va a suceder en la película

Aunque la aplicación se ha planteado como multipropósito, su principal campo de aplicación será el de los mundos virtuales orientados a la educación o el entrenamiento. En este sentido, esta aplicación podría servir como guía al estudiante de cómo realizar alguna tarea de acuerdo con sus objetivos de aprendizaje. Un ejemplo de un mundo virtual que podría sacar partido de la aplicación desarrollada en este trabajo es el de un Laboratorio Virtual de Biotecnología como el presentado en (Riofrío Luzcando, 2012). El NPC en un laboratorio virtual como éste podría mostrar paso a paso la forma correcta de realizar las mezclas químicas, cuando el estudiante necesite asistencia en el proceso de aprendizaje. Así, el estudiante visualizaría lo que realiza el NPC para aprender la forma correcta de realizar dicha tarea.

3.2. Especificación del Plan del NPC en una Ontología

Las acciones que realizará el NPC se van a definir dentro de una ontología que se definirá como una extensión de la ontología del estudiante de Julia. El motivo por el que se ha escogido esta ontología como punto de partida es porque esta ontología es la más completa que conocemos para modelar acciones de estudiantes en un mundo virtual. Además, está previsto que esta aplicación se integre con un SIT que ya utiliza esta ontología y que utilizará los NPCs para proporcionar explicaciones a los estudiantes.

En primer lugar, se van a especificar cada una de las acciones que se pretende que pueda realizar el NPC en el mundo virtual creado con OpenSimulator. Estas acciones son las siguientes:

- *Teletransportación* – coloca al NPC en una posición especificada.
- *Tocar Objeto* – el NPC realiza la acción de tocar un objeto específico.
- *Mensaje* – se utilizará para enviar mensajes a un avatar o a un objeto. Por ejemplo se debe enviar un mensaje por el chat a un avatar o se debe seleccionar una opción del menú (se envía un mensaje al objeto que muestra el menú).
- *Mover* – dirige al NPC hacia la posición indicada.
- *Volar* – cambia el NPC al estado “volando”.
- *Caminar* – cambia al NPC al estado “caminando”.
- *Correr* – cambia al NPC al estado “corriendo”.
- *Tomar Objeto desde el Inventario* – esta acción adjunta el objeto que se encuentra en el inventario del NPC a una parte del cuerpo especificada. Si no se especifica la parte del cuerpo, por defecto se utiliza la mano derecha.
- *Sentarse en un Objeto Cercano* – hace que el NPC se siente en el objeto más cercano del tipo especificado.
- *Sentarse en un Objeto Disponible* – hace sentar al NPC en un objeto disponible del tipo especificado.

- *Sentarse* – sienta al NPC en la posición actual.
- *Girar Hacia* – hace girar al NPC mirando hacia una posición indicada.
- *Tocar Objeto de Avatar Más Cercano* – hace tocar el objeto del tipo dado más cercano al NPC y que pertenece a este avatar.
- *Añadir Objeto al Inventario* – agrega un objeto dado al inventario del NPC.
- *Caminar Cerca de un Objeto Disponible* – dirige el NPC a una posición cercana a un objeto dado que se encuentre disponible, es decir, que no esté siendo utilizado por otro avatar.

Una vez definidos estos tipos de acciones en la ontología, debería ser posible crear instancias de estas acciones en la misma ontología, de forma que puedan ser ejecutadas posteriormente en el mundo virtual. Estas instancias serán definidas de acuerdo a la actividad que se requiere que el NPC realice.

3.3. Especificación del Motor de Ejecución

La ejecución del plan especificado en una ontología (sección 3.2) la realizará la aplicación a desarrollar leyendo el archivo de instancias de la ontología de dicho plan.

Las principales funciones que se deben implementar para que la aplicación pueda llevar a cabo su cometido son las siguientes:

Funciones asociadas con el tratamiento de la ontología con las acciones

- *Leer Ontología de Acciones* – es la encargada de leer los datos de configuración de la ontología para cargar la ontología con las definiciones de las acciones.
- *Cargar Plan de acción del NPC* – se encarga de cargar en memoria las instancias de las acciones que definen el plan a realizar por el NPC.
- *Consultas a la Ontología mediante sentencia SPARQL* – traerá los resultados de la ontología de acuerdo a la sentencia SPARQL enviada.
- *Crear consulta SPARQL para consultar una acción específica* – será la encargada de construir las consultas para cada acción de la ontología debido a que cada acción tiene su estructura específica.
- *Obtener Acción n-ésima* – obtiene el nombre de la acción que ocupa la n-ésima posición en el plan a realizar.

Funciones asociadas con la interacción del NPC con el Mundo Virtual

- *Autentificar con el Mundo Virtual* – se encarga de establecer la conexión con el mundo virtual para poder controlar a un NPC ya existente en el mundo virtual.

- *Enviar Mensaje* – esta acción permite enviar un mensaje a un objeto del mundo virtual.
- *Petición para Sentarse en un Objeto* – envía al mundo virtual una solicitud para sentar al NPC en un objeto específico.
- *Tomar Objeto del inventario* – adjunta un objeto que se encuentra en el inventario a una parte del cuerpo del NPC.
- *Dejar Objeto adjuntado en el inventario* – quita el objeto que se encuentra en una parte del cuerpo del avatar y lo deja en el inventario.
- *Agregar Objeto a Inventario* – agrega un objeto al inventario del NPC.
- *Petición de Propiedades de un Objeto* – solicita al servidor del mundo virtual, propiedades de un objeto dado tales como nombre, posición, etc.
- *Buscar el Objeto del Avatar más cercano* – se encarga de traer el objeto que se encuentre más cercano al NPC y que pertenece a este avatar.
- *Obtener del Objeto más cercano* – devuelve el objeto más cercano al NPC.
- *Buscar Objeto* – devuelve un objeto en base a criterios de búsqueda.
- *Correr* – cambia al NPC al estado "corriendo".
- *Sentarse* – hace que el NPC se siente en el lugar donde se encuentra actualmente.
- *Pararse* – hace que el NPC se ponga en posición de pie.
- *Teletransportación* – Coloca el NPC en una posición determinada.
- *Tocar Objeto* – toca un objeto específico.
- *Tocar Objeto más Cercano* – toca el objeto más próximo al avatar.
- *Caminar* – cambia al estado de "caminando" al NPC.
- *Caminar hacia la Posición más Cercana de un Objeto* – dirige el avatar hacia la posición más próxima disponible de un objeto.
- *Dejar de Volar* – detiene el estado de "volar" del NPC.
- *Sentarse en el suelo* – hace que el NPC se siente en el suelo.
- *Dejar de Moverse* – hace que el NPC se detenga.
- *Girar hacia la izquierda o derecha* – hace que el NPC empiece a girar hacia su izquierda o su derecha.
- *Dejar de girar a la izquierda o derecha* – detiene el giro hacia la izquierda o derecha del NPC.
- *Mover hacia una Coordenada Específica* – hacer mover el avatar en el estado que se encuentra, ya sea caminando, corriendo o volando, hacia una coordenada específica en un plano *X, Y, Z*.
- *Dejar de moverse en una coordenada específica* – detiene el movimiento del avatar en una coordenada específica.

Capítulo 4

Desarrollo

En este capítulo se detalla cómo se implementó la aplicación BotManager por medio del modelado de las acciones en una ontología definida como una extensión de una ontología de estudiante ya existente, y la implementación del Motor de Ejecución de las acciones. La implementación finalmente se realizó en C#, ya que la librería LibOpenMetaverse, necesaria para controlar el NPC de OpenSimulator, está implementada en C#.

La figura 4.1 muestra el esquema de la solución adoptada. La aplicación se enfoca en dos acciones principales que son:

1. Leer la ontología del estudiante de Julia.
2. Ejecutar las acciones que se leyeron de la ontología y ejecutarlas en el mundo virtual por medio del NPC.

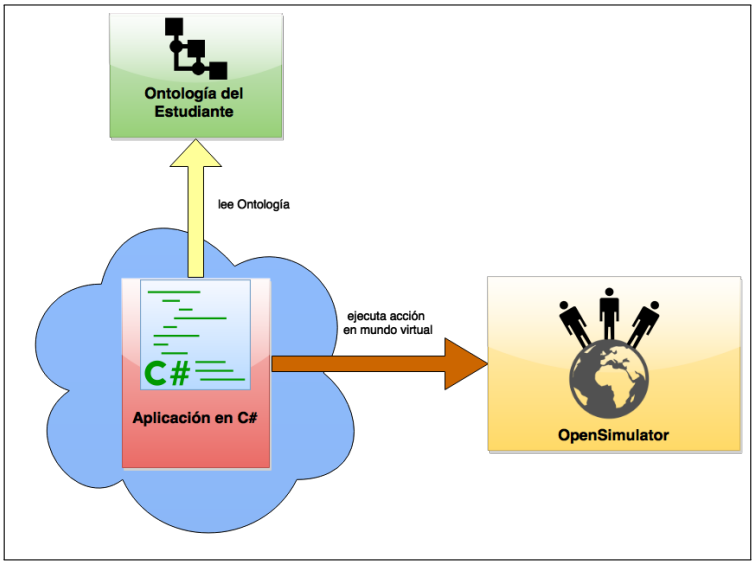

Figura 4.1: Esquema de la arquitectura de BotManager desarrollado en C#

4.1. Modelado de las Acciones en una Ontología

4.1.1. Ontología del Estudiante

El BotManager se implementa para poder conectarse con la ontología del estudiante presentada en el trabajo (Párraga, 2011) (de ahora en adelante la llamaremos *Ontología de Julia*). En este trabajo la Ontología de Julia se engloba en un modelo del estudiante que posee las siguientes características:

- Propone un SIT con un mecanismo de Modelado del Estudiante que aplica técnicas de Ingeniería Ontológica y tiene en cuenta principios pedagógicos.
- Contiene un modelo de datos respecto al estudiante, extenso y fácilmente extensible y acomodable a las necesidades requeridas para diversos SITs y sistemas de aprendizaje.
- Define un método de diagnóstico pedagógico y cognitivo basado en razonamiento no monótono, que tiene en cuenta las interacciones del estudiante con el mundo y otros aspectos registrados en el modelo de datos.

La figura 4.2 muestra la arquitectura del modelo del estudiante propuesto, dividida en cuatro partes fundamentales:

1. *Diseño del plan de estudios de la materia de enseñanza* – establece las actividades dentro del proceso de aprendizaje del estudiante, así como las metas que éste requiere alcanzar.
2. *Módulo del Experto* – establece las acciones que se pueden realizar para concluir una actividad exitosamente.
3. *Ejecución de Alguna Acción* – cuando el estudiante ejecuta una acción se registra en la ontología.
4. *Diagnóstico Pedagógico y Cognitivo* – el primer tipo de diagnóstico utiliza una serie de reglas de diagnóstico para establecer los objetivos de aprendizaje alcanzados o no por el estudiante cuando ha ejecutado una acción. El segundo deduce cuál es el nivel de conocimientos del estudiante en base al cumplimiento de objetivos obtenido.

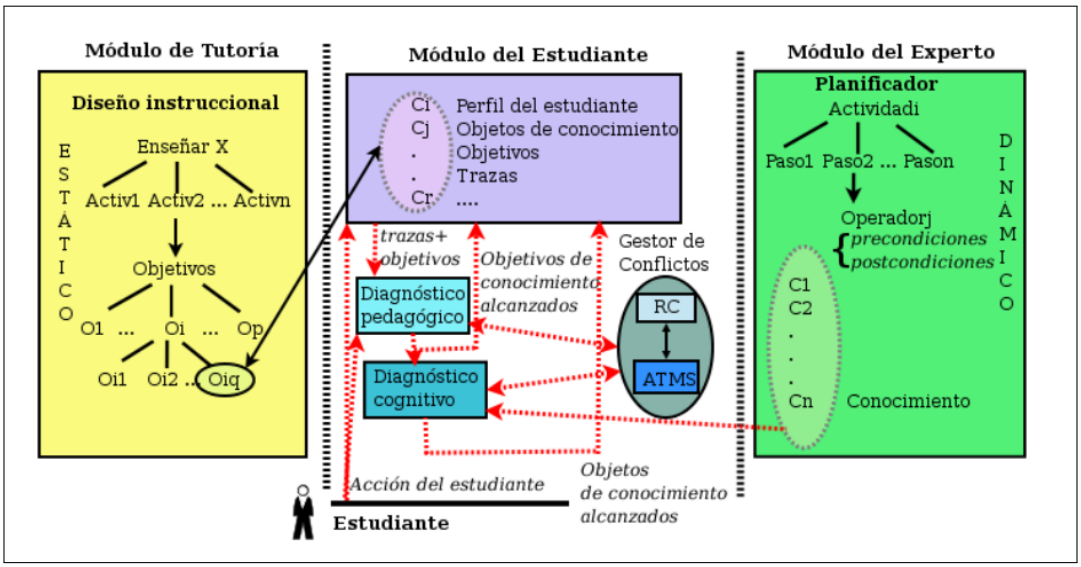

Figura 4.2: Arquitectura del modelo del estudiante. Fuente: (Párraga, 2011)

La ontología del estudiante está dividida en las ontologías modulares que se puede ver en la figura 4.3.

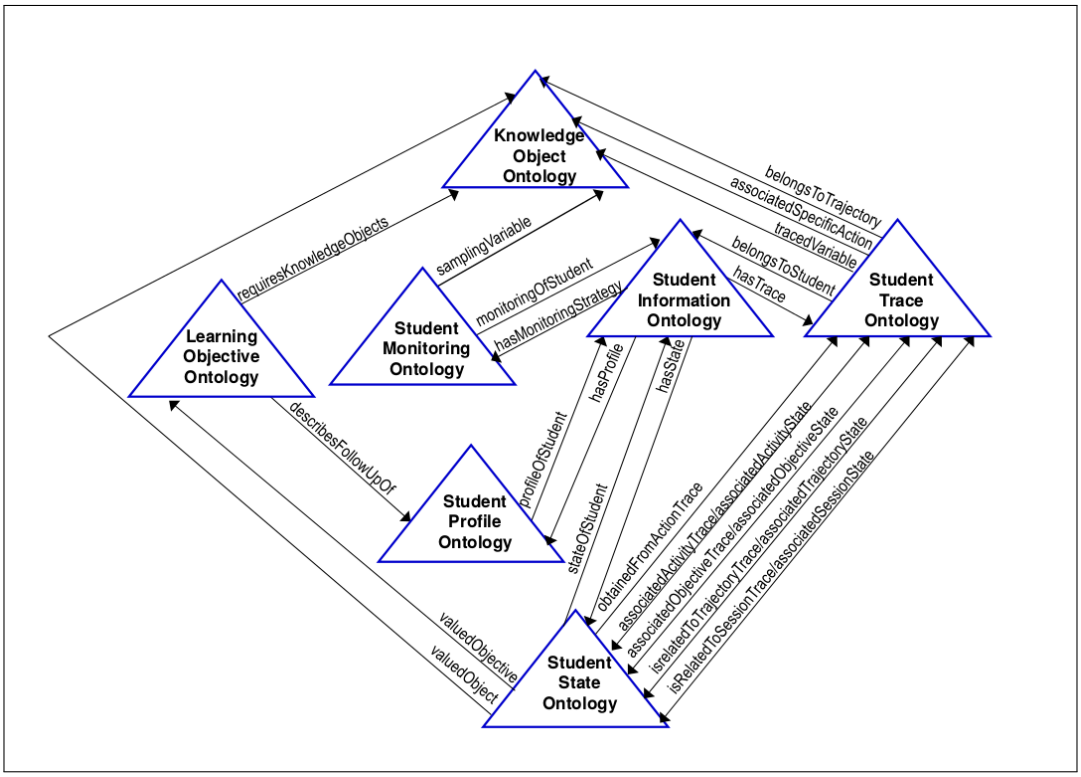

Figura 4.3: Estructura modular de la ontología del estudiante. Fuente: (Párraga, 2011)

1. *Ontología Student_Profile* – contiene la información personal del estudiante, tales como datos personales, rasgos físicos, perfil psicológico, preferencias de interacción, etc.

2. *Ontología Student_State* – describe el estado actual de los conocimientos del estudiante, estado emocional, estado pedagógico, estado de capacidades generales y competencias.

3. *Ontología Student_Trace* – mantiene el registro de cada actividad del estudiante durante el aprendizaje. De aquí se obtiene la información para la ontología *Student_State*. Además, registra los cambios en los estados de las objetivos de aprendizaje durante el tiempo.

4. *Ontología_Student_Information* – contiene el concepto raíz de toda la información del estudiante, y para ello utiliza la información de las ontologías *Student_Profile*, *Student_State* y *Student_Trace*.

5. *Ontología Learning_Objetive* – aquí se especifican objetivos de distintos dominios, tales como el afectivo, el psicomotriz y el cognitivo, y con distintos niveles de abstracción.

6. *Ontología_Student_Monitoring* – se describen las características del método de seguimiento de algunas variables en el aprendizaje.

7. *Knowledge_Object* – almacena los objetos de conocimiento del modelo del estudiante. Está clasificada en los objetos que describen un conocimiento estructural *Structural_Knowledge* (véase la figura 4.4 y objetos que describen un conocimiento procedimental *Procedural_-Knowledge*(véase la figura 4.5).

Las cuatro primeras ontologías son dependientes del estudiante, mientras que las tres últimas son independientes del estudiante, ya que se pueden definir instancias de conceptos, que pueden ser compartidas por varios estudiantes.

Figura 4.4: Jerarquía de *Structural_Knowledge* en la ontología *Knowledge_Object*. Fuente: (Párraga, 2011)

Figura 4.5: Jerarquía de *Procedural_Knowledge* en la ontología *Knowledge_Object*. Fuente: (Párraga, 2011)

La figura 4.6 muestra de forma general los conceptos más relevantes de las ontologías, así como las principales relaciones entre ellos.

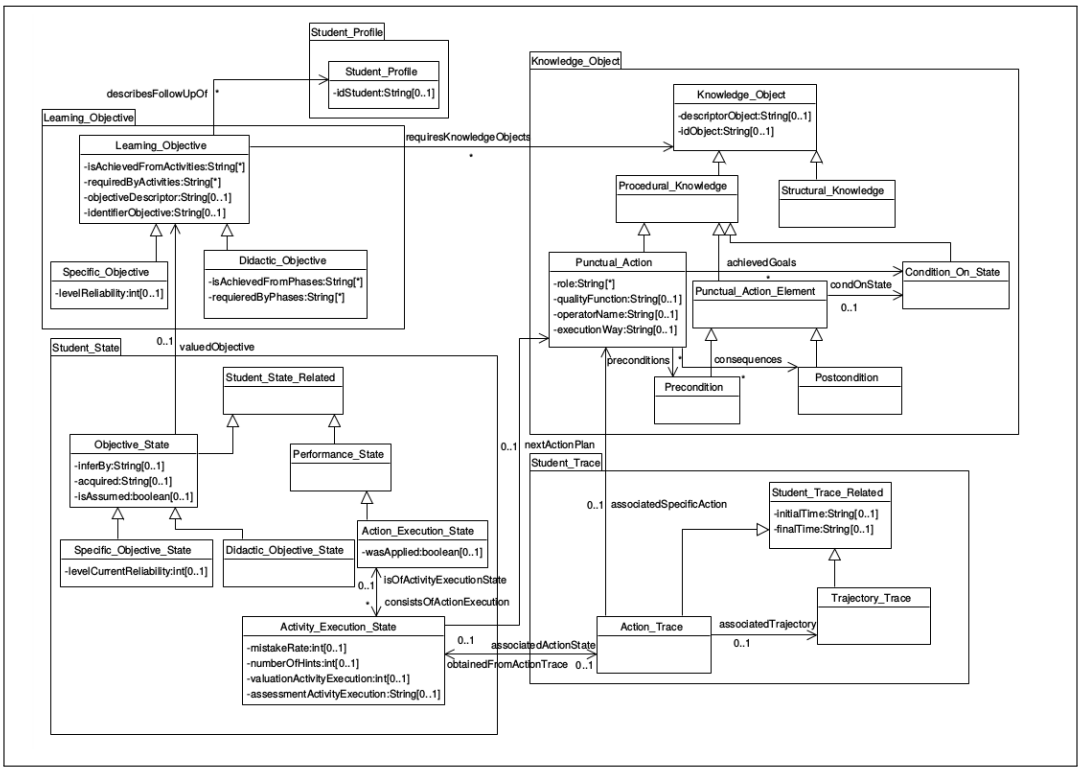

Figura 4.6: Visión general de la ontología del estudiante. Fuente: (Párraga, 2011)

4.1.2. Modelado de Acciones del NPC en la Ontología del Estudiante

Para especificar las acciones que realizará el NPC, se debe establecer un plan de acciones. El plan de acciones se encuentra modelado mediante la clase *Plan* perteneciente a la ontología *Knowledge_Object* (ver figura 4.4). Lo siguiente es la especificación de cada acción en particular y el orden de ejecución de esta; esto se define con la clase *Plan_Element* (ver figura 4.4).

Las clases de acciones listadas en la sección 3 que no se encuentran en la ontología *Knowledge_Object* necesarias para controlar el NPC en el mundo virtual, se definen en una nueva ontología llamada *Knowledge_Object_Bot_Manager*, que extiende *Knowledge_Object*.

Las nuevas clases creadas en *Knowledge_Object_Bot_Manager* extienden las siguientes clases ya existentes en *Knowledge_Object*: *Modifies_Student_Position* (figura 4.7), *Modifies_Relation_Student_Object* (veáse la figura 4.8), *Not_Verbal* (véase la figura 4.9), *Simple_Object* (véase la figura 4.10), *Geometric_Object* (véase la figura 4.11) y *Concept* (véase la figura 4.12).

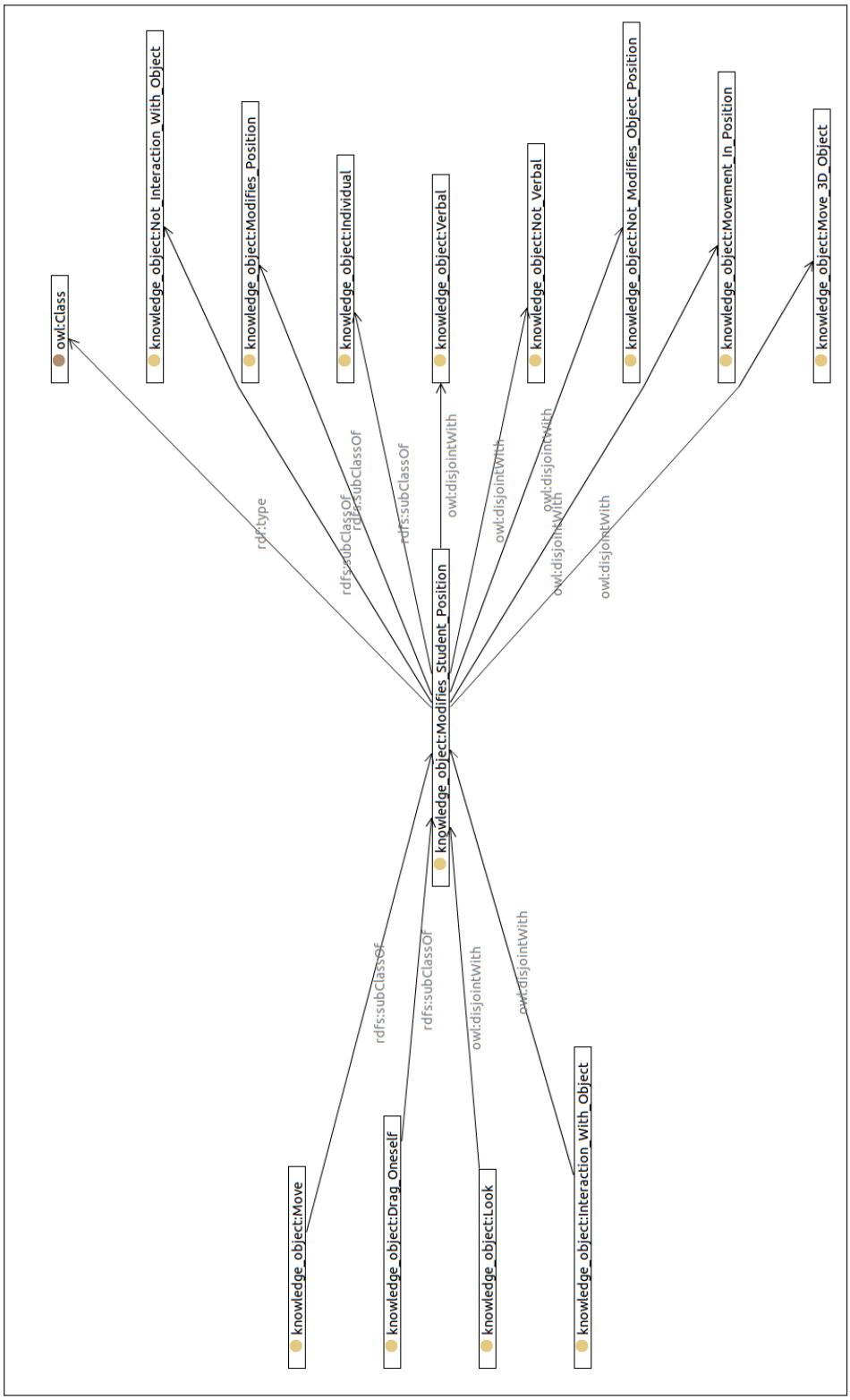

Figura 4.7: Clase Modifies_Student_Position. Fuente: (Párraga, 2011)

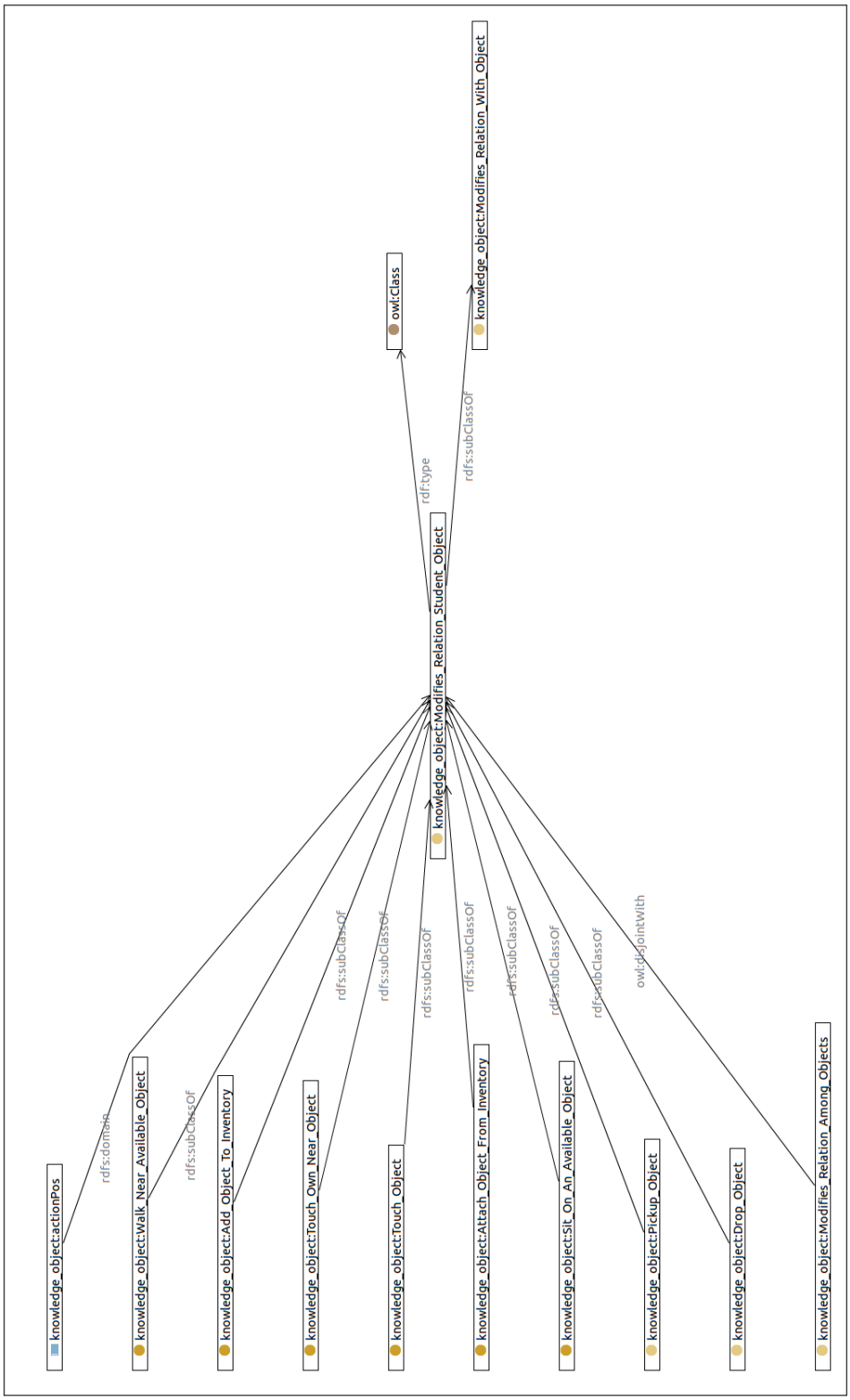

Figura 4.8: Clase Modifies_Relation_Student_Object. Fuente: (Párraga, 2011)

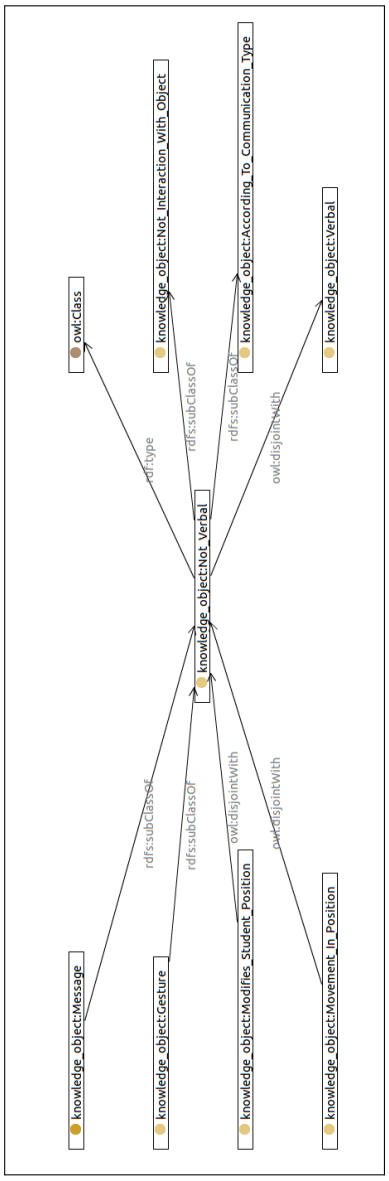

Figura 4.9: Clase Not_Verbal. Fuente: (Párraga, 2011)

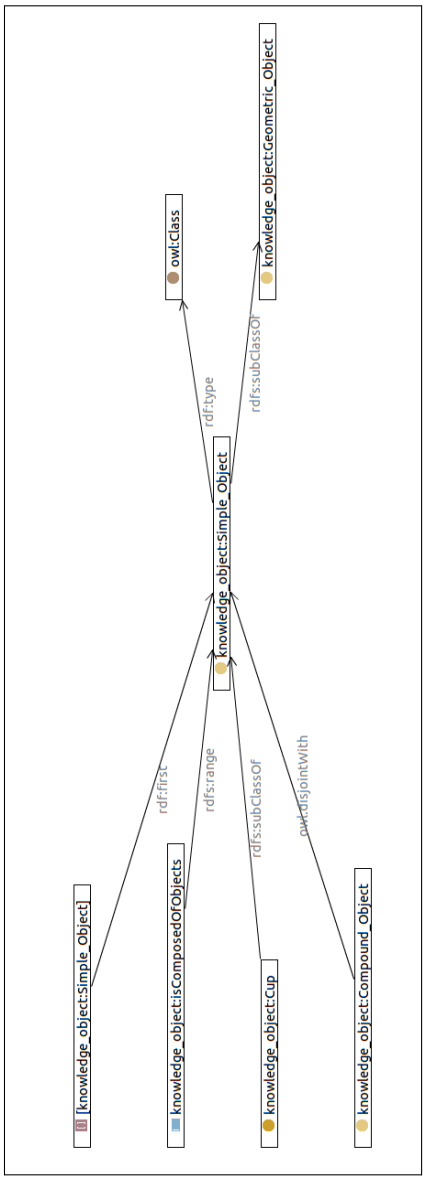

Figura 4.10: Clase Simple_Object. Fuente: (Párraga, 2011)

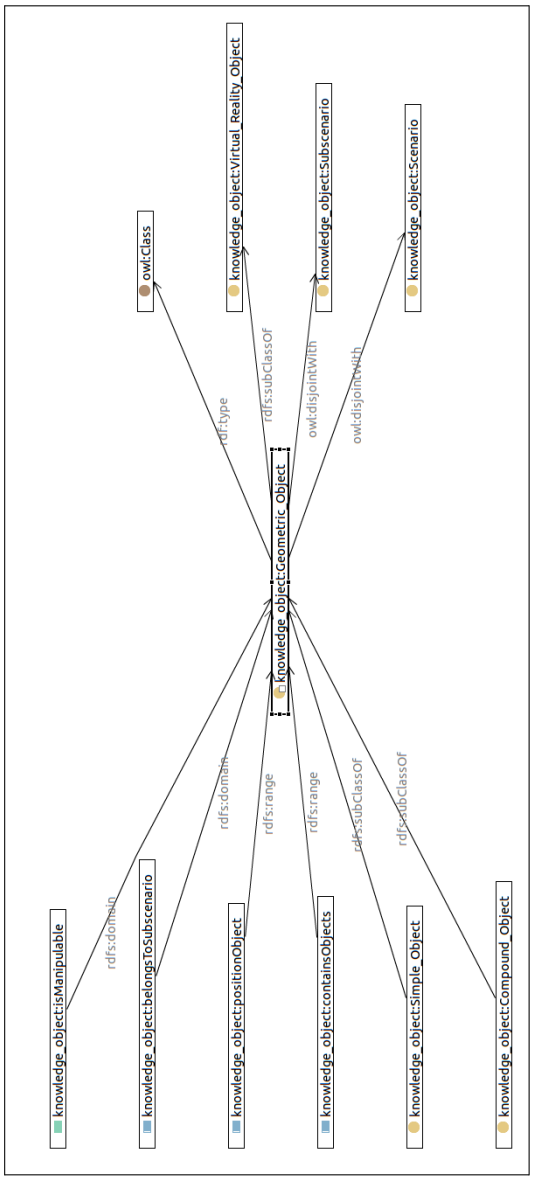

Figura 4.11: Clase Geometric_Object. Fuente: (Párraga, 2011)

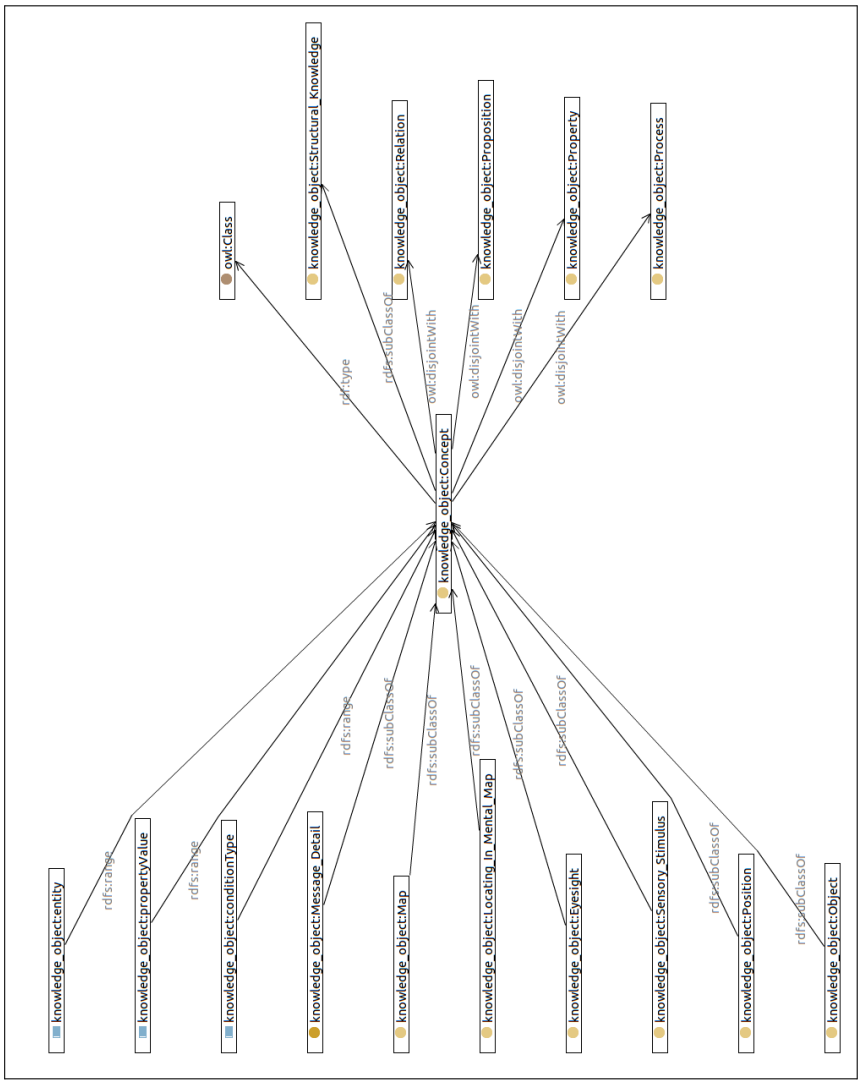

Figura 4.12: Clase Concept. Fuente: (Párraga, 2011)

4.1.2.1. Listado de Nuevas Acciones

En este apartado se va a detallar cómo se han definido cada una de las nuevas acciones a partir de la jerarquía de acciones ya existentes en la ontología *Knowledge_Object*.

Tomar Objeto desde Inventario

Clase	**Attach_Object_From_Inventory**.
Descripción	Describe la acción que implica adjuntar el objeto que se encuentra en el inventario del NPC a una parte del cuerpo, por defecto la mano derecha.
Superclase	*Modifies_Relation_Student_Object* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObject (Párraga, 2011).

Tabla 4.1: Clase Attach_Object_From_Inventory

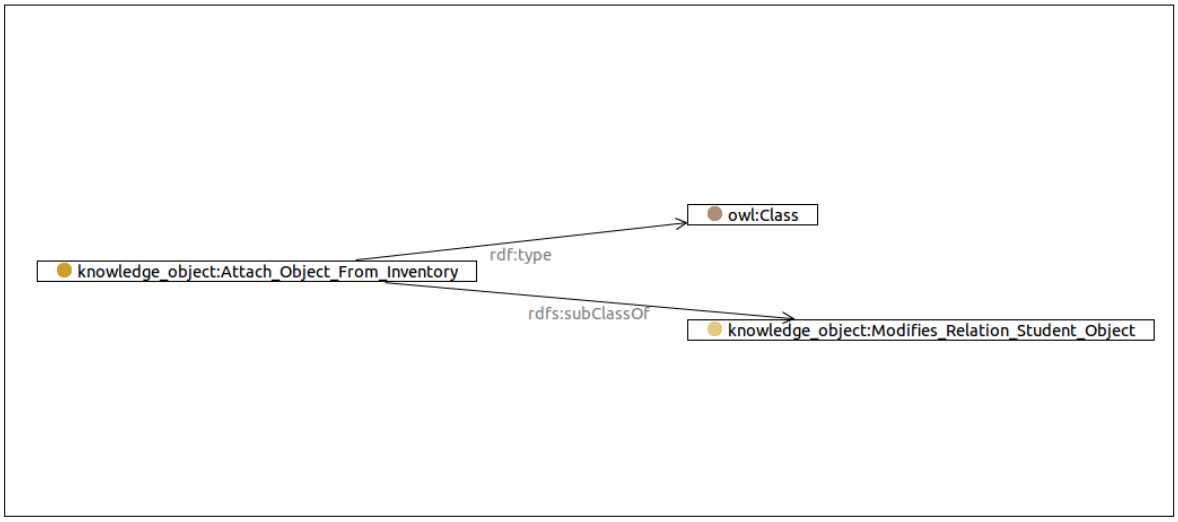

Figura 4.13: Clase Attach_Object_From_Inventory

Tocar Objeto

Clase	**Touch_Object**.
Descripción	Describe la acción que implica tocar un objeto.
Superclase	*Modifies_Relation_Student_Object*.
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObjects (Párraga, 2011).

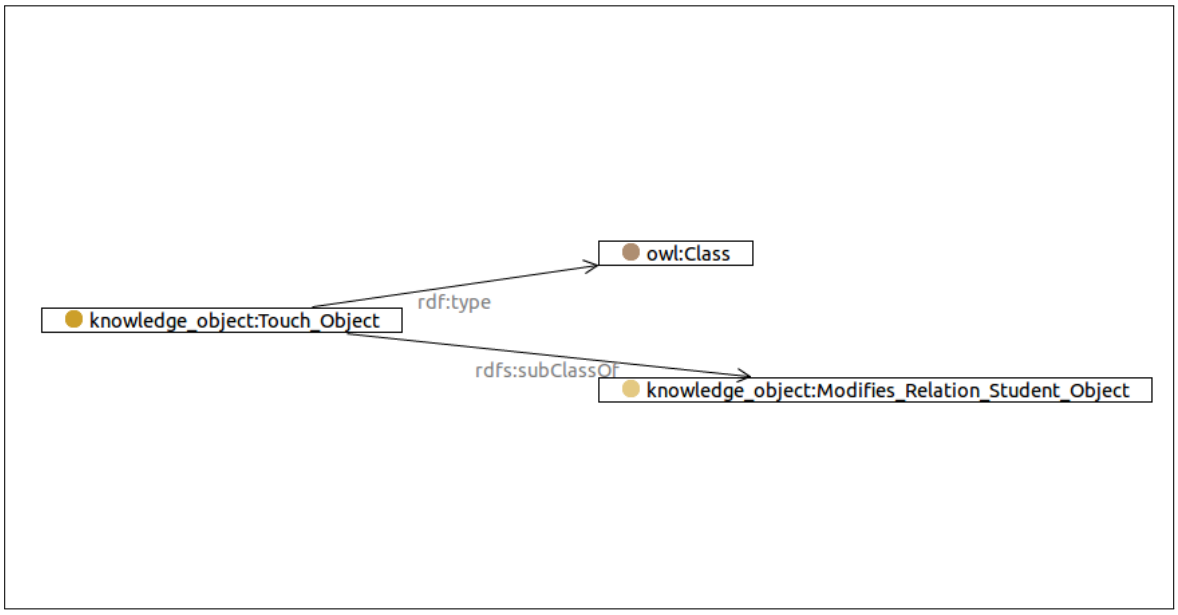

Figura 4.14: Clase Touch_Object

Teletransportación

Clase	**Teleport**.
Descripción	Describe la acción que implica dirigir el avatar inmediatamente a una posición especifica.
Superclase	*Move* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	destinationPos (Párraga, 2011).

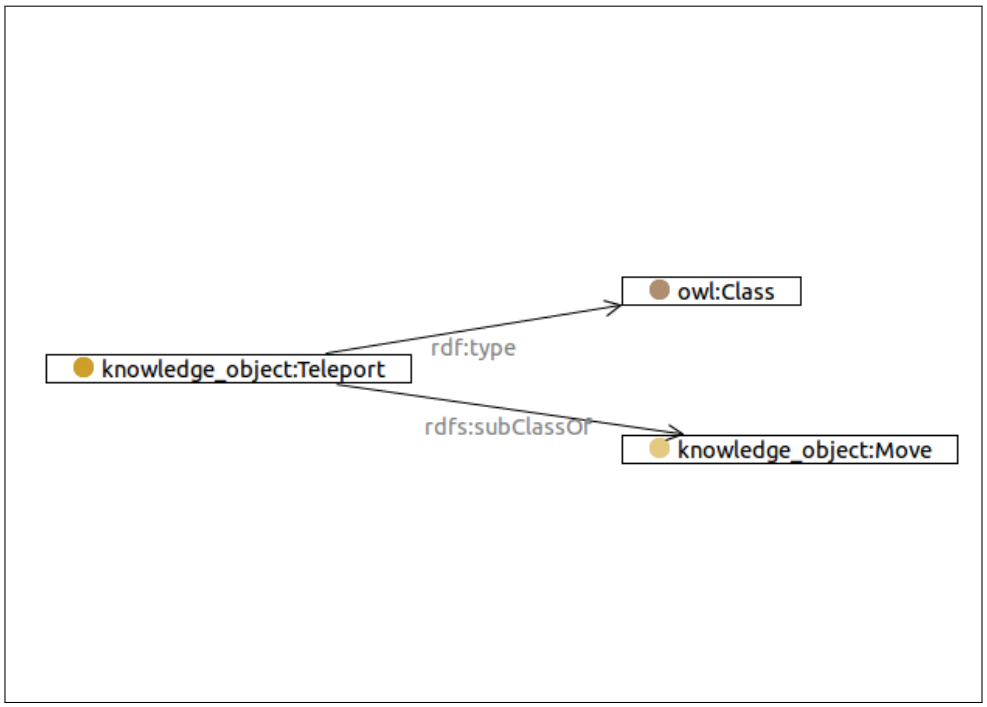

Figura 4.15: Clase Teleport

Mensaje a Mundo Virtual

Clase	**Message**.
Descripción	Describe la acción que implicar mandar un mensaje a un objeto del mundo virtual. Por ejemplo, es usado cuando en el mundo virtual, el NPC tiene que escoger entre varias opciones asociadas a distintas acciones que se le muestran en un menú. Al mandar el mensaje es necesario especificar el canal del mundo virtual por el que se va a mandar.
Superclase	*Not_Verbal* (Párraga, 2011).
Propiedades Definidas	*chatToSimulator*: define un mensaje de chat para enviar al mundo virtual.
Propiedades Heredadas Utilizadas	

Tabla 4.2: Clase Message

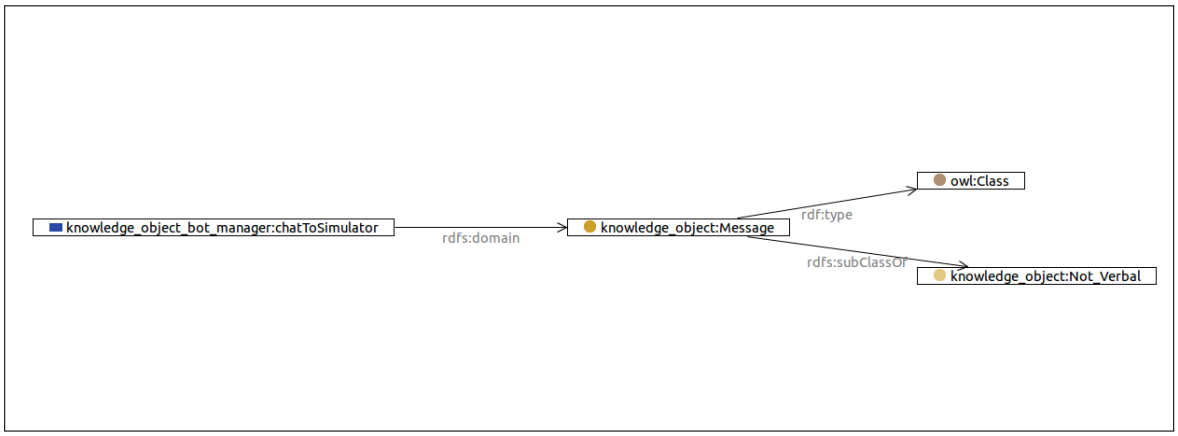

Figura 4.16: Clase Message

Volar

Clase	**Fly**.
Descripción	Describe la acción que implica que el NPC cambie su estado a volar.
Superclase	*Move* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	

Tabla 4.3: Clase Fly

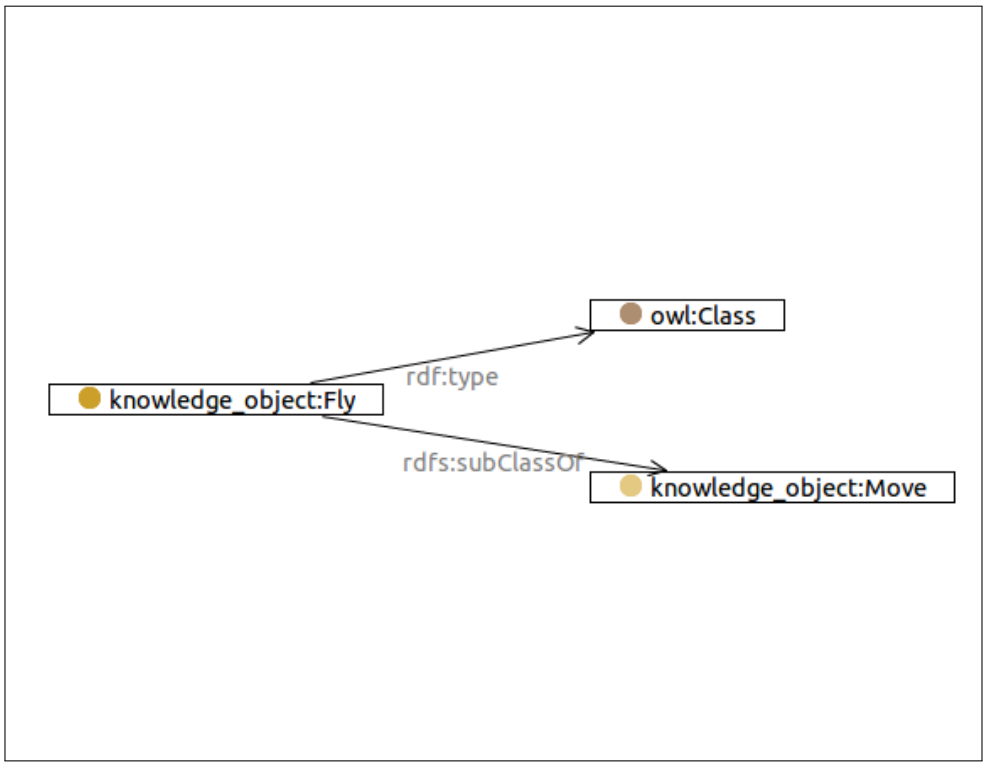

Figura 4.17: Clase Fly

Correr

Clase	**Run**.
Descripción	Describe la acción que implica que el avatar cambie su estado a correr.
Superclase	*Move* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	

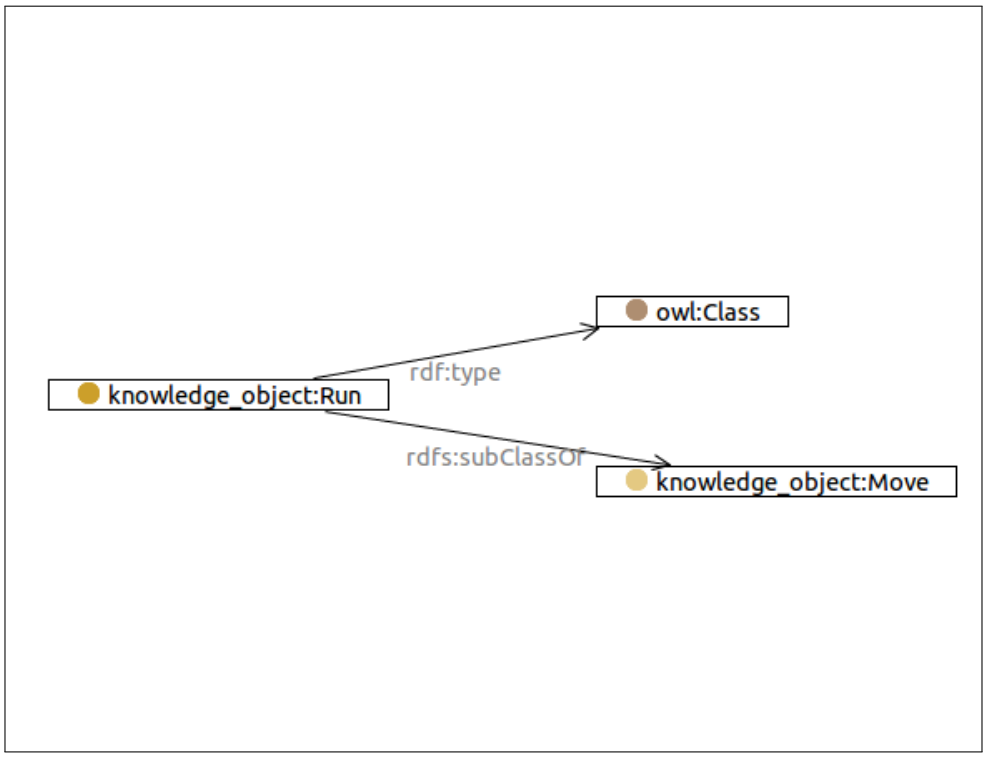

Figura 4.18: Clase Run

Girar Hacia

Clase	**Turn_Toward**.
Descripción	Describe la acción que implica que el avatar gire hacia la posición indicada.
Superclase	*Movement_In_Position* (Párraga, 2011).
Propiedades Definidas	*facedObject*: especifica el objeto hacia el que girará el avatar.
Propiedades Heredadas Utilizadas	

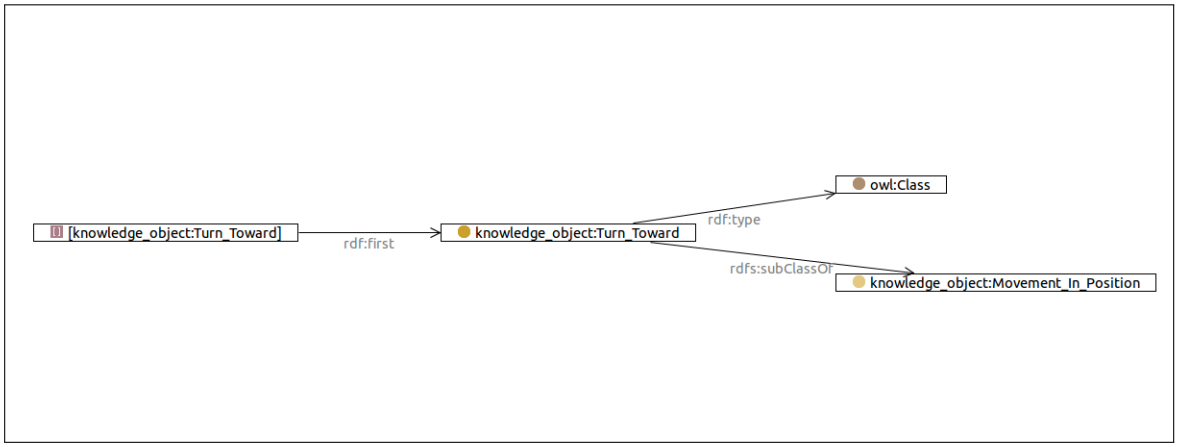

Figura 4.19: Clase Turn_Toward

Caminar

Clase	**Walk**.
Descripción	Describe la acción que implica mover el avatar caminando a una posición específica.
Superclase	*Move* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	*destinationPos* (Párraga, 2011).

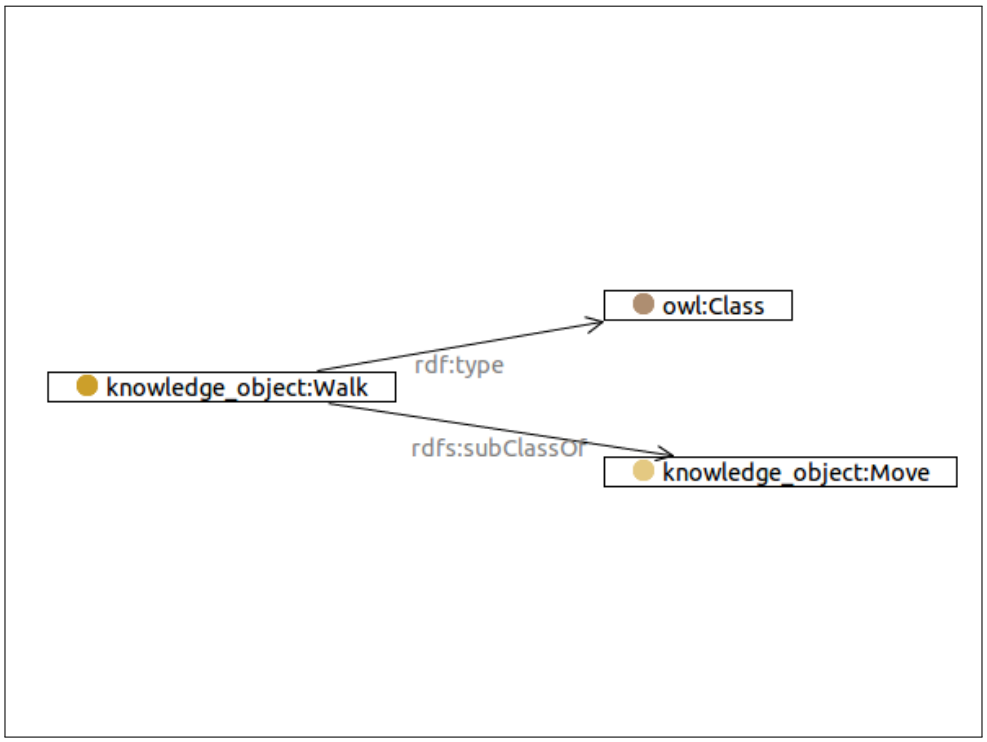

Figura 4.20: Clase Walk

Tocar Objeto Más Cercano Propio

Clase	**Touch_Own_Near_Object**.
Descripción	Describe la acción que implica tocar el objeto más próximo al NPC perteneciente a él.
Superclase	*Modifies_Relation_Student_Object* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObjects (Párraga, 2011).

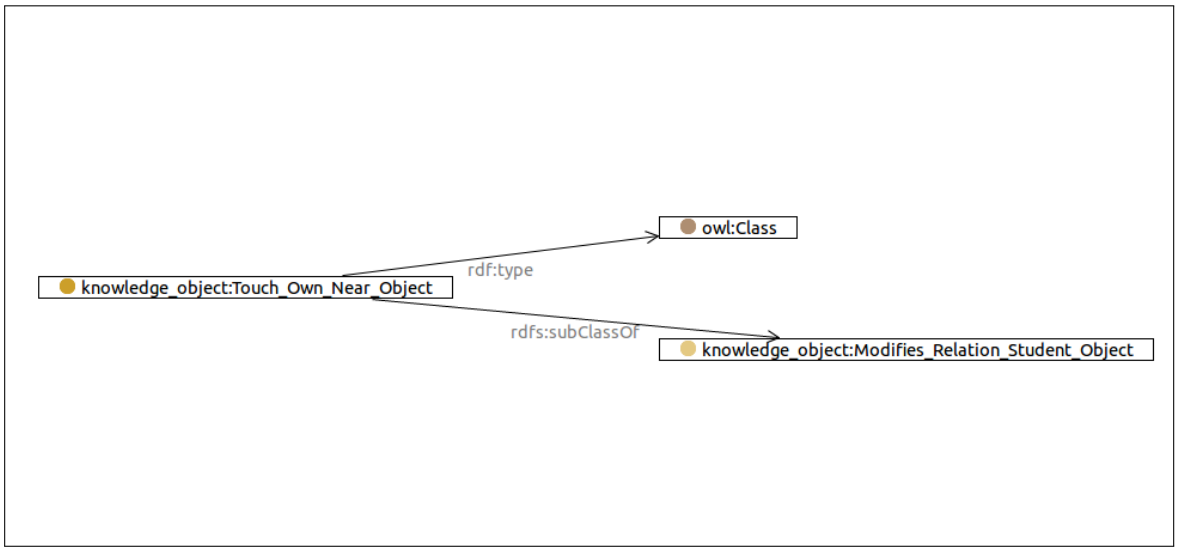

Figura 4.21: Clase Touch_Own_Near_Object

Añadir Objeto a Inventario

Clase	**Add_Object_To_Inventory**.
Descripción	Describe la acción que implica agregar un objeto al inventario del NPC.
Superclase	*Modifies_Relation_Student_Object* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObjects (Párraga, 2011).

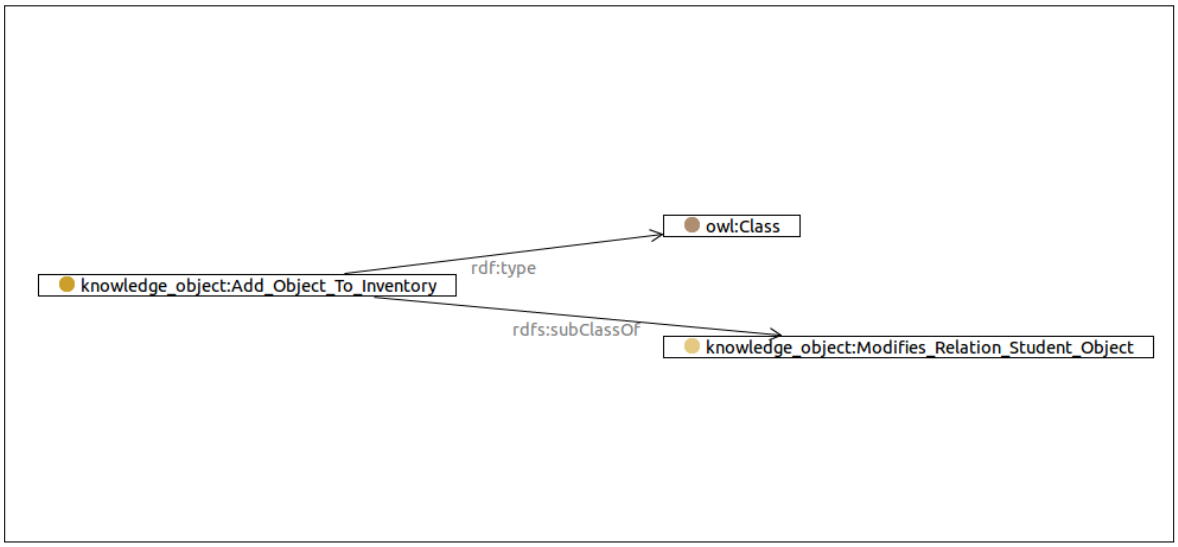

Figura 4.22: Clase Add_Object_To_Inventory

Caminar Cerca de un Objeto Disponible

Clase	**Walk_Near_Available_Object**.
Descripción	Describe la acción que implica movilizar el avatar caminando hacia un objeto disponible.
Superclase	*Modifies_Relation_Student_Object* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObjects (Párraga, 2011).

Tabla 4.4: Clase Walk_Near_Available_Object

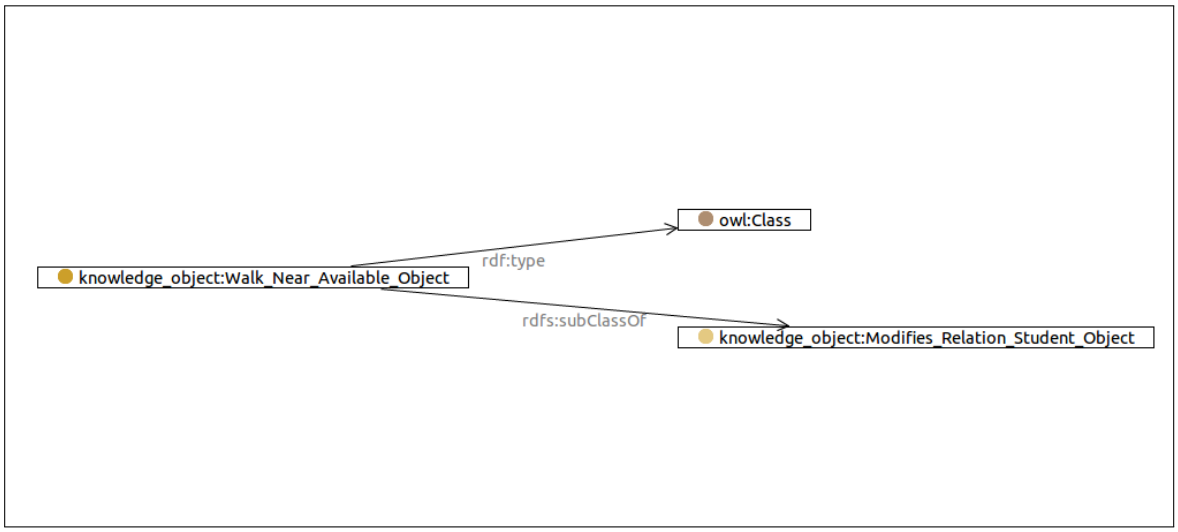

Figura 4.23: Clase Walk_Near_Available_Object

Sentarse

Clase	**Sit**.
Descripción	Describe la acción que implica sentarse en el suelo.
Superclase	*Movement_In_Position*.
Propiedades Definidas	
Propiedades Heredadas Utilizadas	

Tabla 4.5: Clase Sit

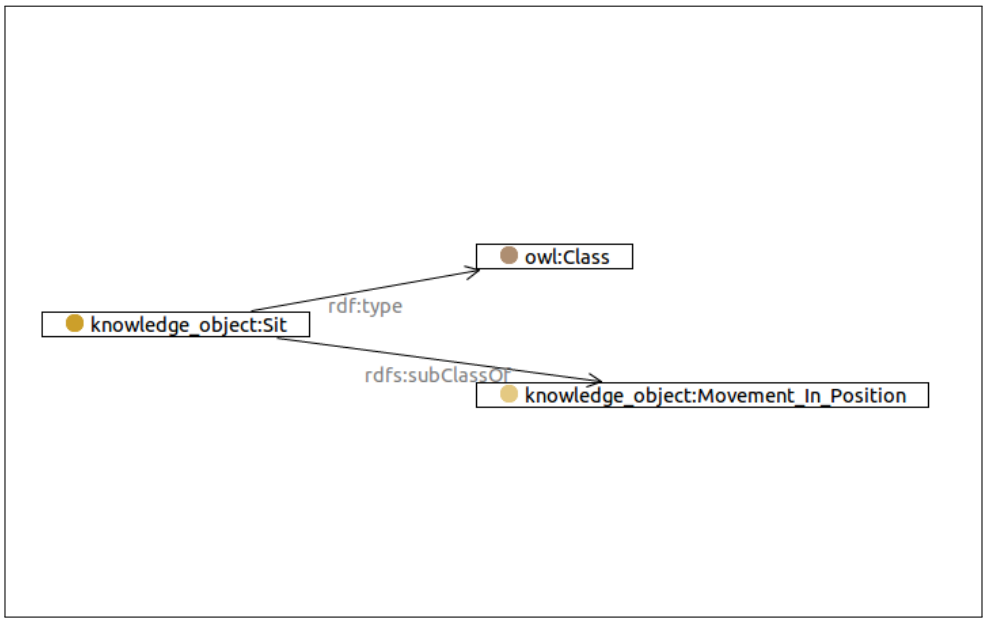

Figura 4.24: Clase Sit

Sentarse en un objeto disponible

Clase	**Sit_On_An_Available_Object**.
Descripción	Describe la acción que implica sentarse en un objeto disponible.
Superclase	*Modifies_Relation_Student_Object* (Párraga, 2011).
Propiedades Definidas	
Propiedades Heredadas Utilizadas	isAppliedToObjects (Párraga, 2011)

Tabla 4.6: Clase Sit_On_An_Available_Object

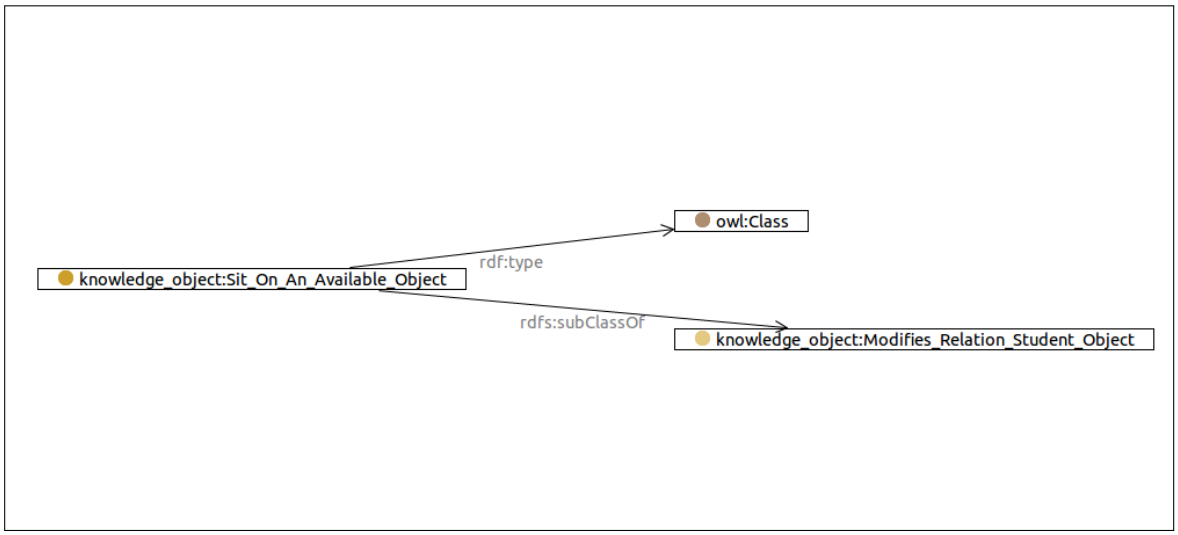

Figura 4.25: Clase Sit_On_An_Available_Object

4.1.2.2. Listado de Nuevos Objetos para las Acciones

Taza

Clase	**Cup**.
Descripción	Es el objeto específico taza.
Superclase	*Simple_Object* (Párraga, 2011).
Propiedades Definidas	*name*: define el nombre del objeto.
Propiedades Heredadas Utilizadas	

Tabla 4.7: Clase Cup

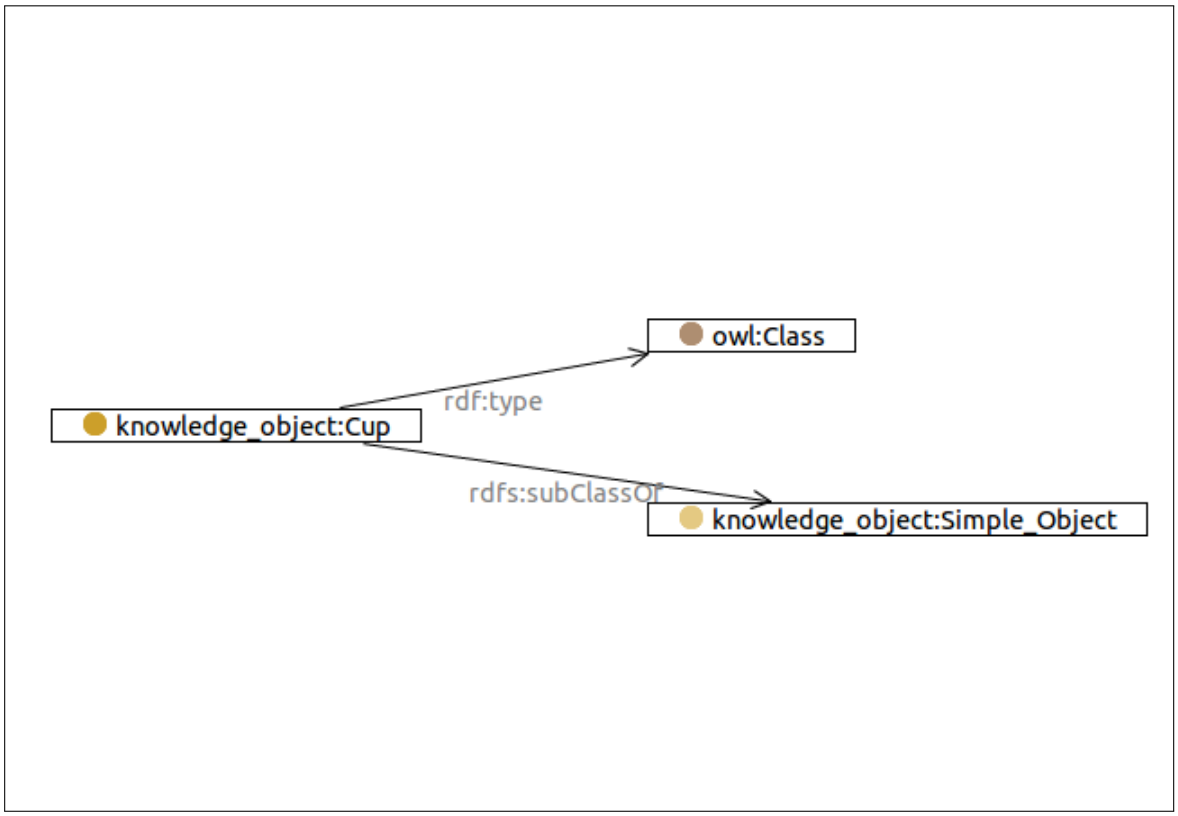

Figura 4.26: Clase Cup

Detalle del Mensaje

Clase	**Message_Detail**.
Descripción	Describe el objeto que detalla el mensaje que será enviado al mundo virtual.
Superclase	*Concept* (Párraga, 2011).
Propiedades Definidas	*channel*: especifica el canal de envío del mensaje, *type*: especifica el tipo de mensaje, *message*: especifica el cuerpo del mensaje.
Propiedades Heredadas Utilizada	

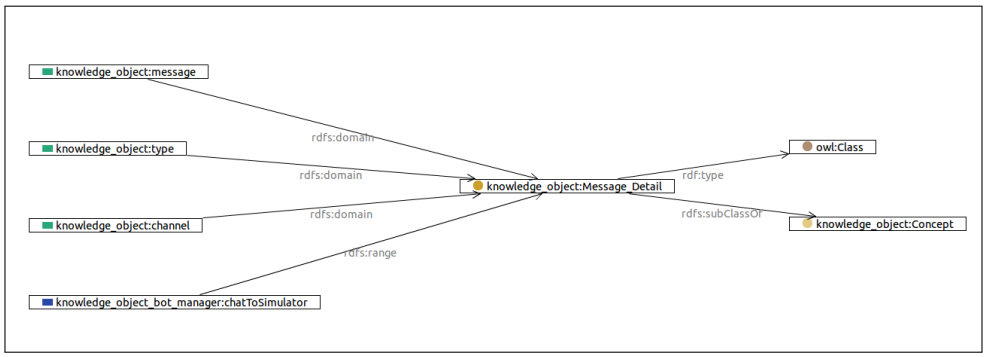

Figura 4.27: Clase Message_Detail

4.1.2.3. Modificaciones a Acciones ya Existentes

Objeto Simple

Clase	**Simple_Object** (Párraga, 2011).
Propiedades Agregadas	*name*: define el nombre del objeto.

Tabla 4.8: Clase Simple_Object

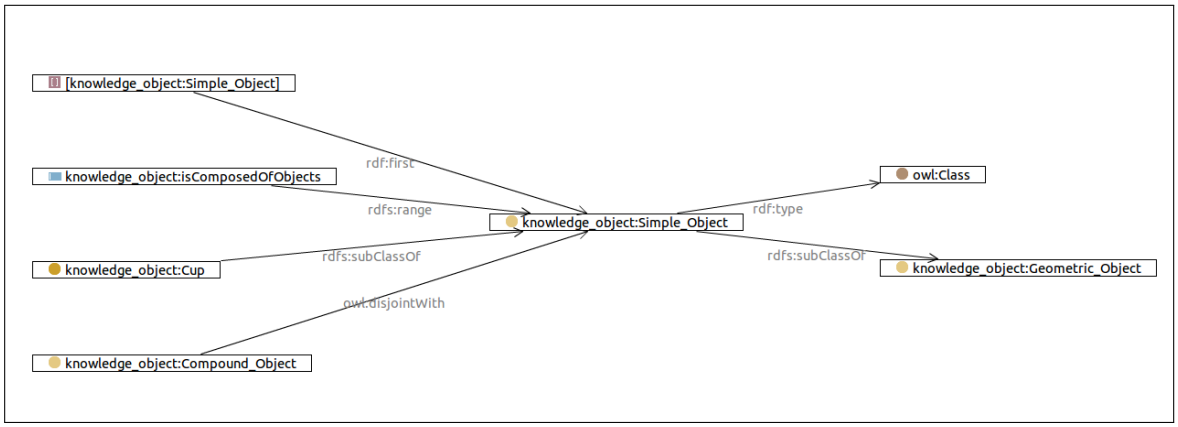

Figura 4.28: Clase Simple_Object

4.2. Motor de Ejecución de las Acciones

4.2.1. Arquitectura del BotManager

El proyecto BotManager se encuentra dividido en cinco paquetes. Cada paquete encapsula las clases de acuerdo a la funcionalidad y los objetos que representan. Los paquetes se enumeran a continuación:

- *Avatar*: Contiene las clases de objetos del mundo virtual y las clases para conectarse al mundo virtual.
- *Executor*: Contiene la clase principal de la aplicación, encargada de construir las acciones y las clases utilitarias para conseguir este objetivo.
- *OntologyAccess*: Contiene las clases encargadas de conectarse a la ontología y las clases complementarias para este objetivo.
- *Util*: Contiene las clases utilitarias para toda la aplicación.
- *Actions*: Contiene las interfaces de conocimiento y avatar, las clases de acciones que implementan estas interfaces, así como las clases abstractas para la construcción de clases mediante *reflexión*[1] con los métodos de lectura de datos (AssembleKnowledgeAction) y para la ejecución de las acciones del avatar (ExecuteAvatarAction).

4.2.2. Diseño Detallado

4.2.2.1. Diagramas de Clases

En los diagramas de clases de cada paquete que se muestran a continuación se van a incluir clases de otros paquetes para facilitar la comprensión del diseño. Las clases de otros paquetes se van a distinguir porque sus nombres van a venir precedidos por prefijos (*nombrePaquete::*) que indiquen sus paquetes de procedencia.

[1] es la capacidad que tiene un programa para observar y opcionalmente modificar su estructura de alto nivel

Paquete Avatar

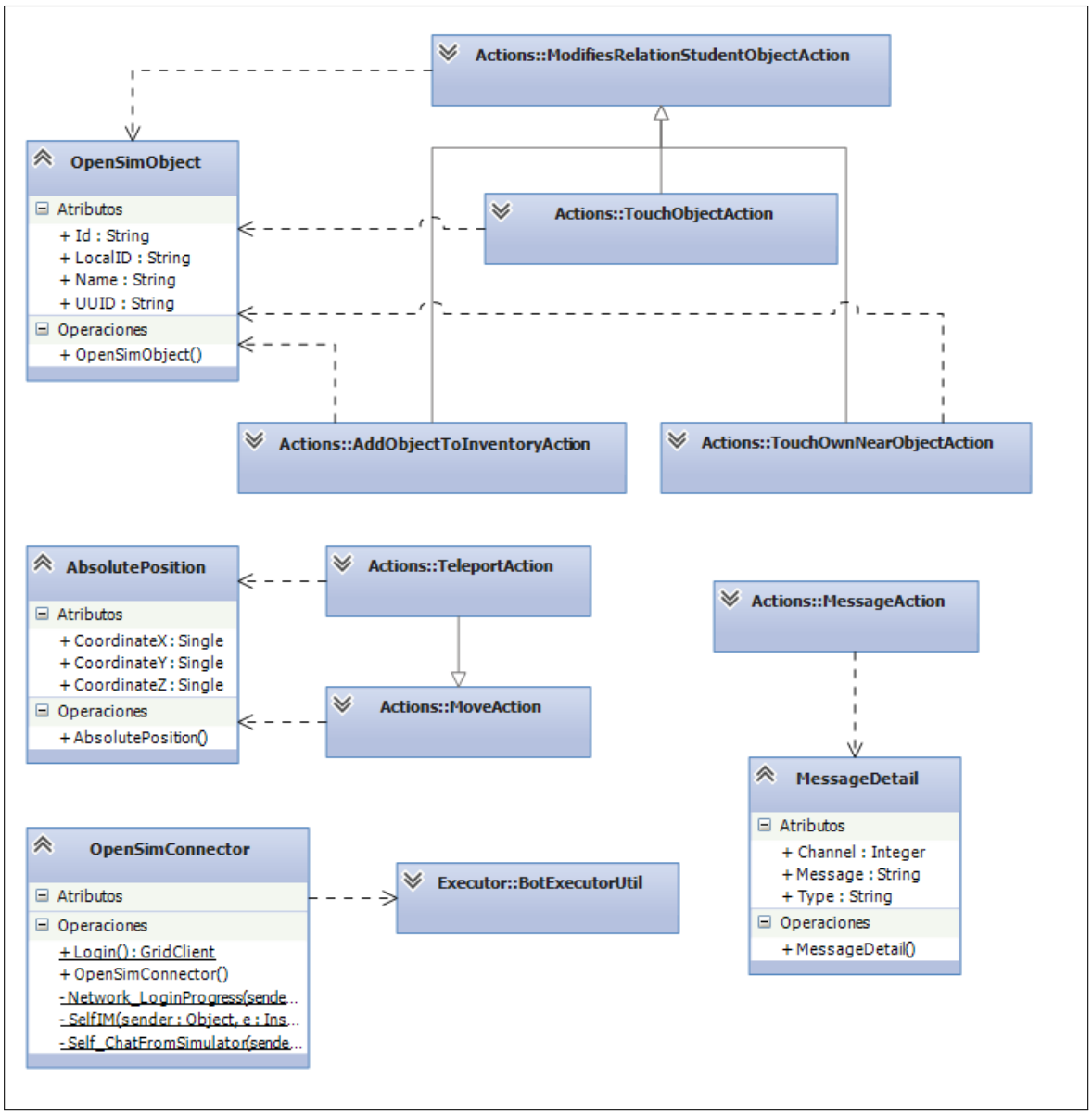

Figura 4.29: Diagrama de Clases de Paquete Avatar

En este paquete se encuentran definidas las clases:

- *AbsolutePosition*: esta clase almacena las coordenadas *X*, *Y Z* que puede tener un avatar o un objeto.
- *AvatarActionsEnum*: es una clase tipo *enum*, contiene las acciones que puede realizar el avatar dentro del mundo virtual.

- *MessageDetail*: esta clase almacena el detalle de un mensaje que puede ser enviado al mundo virtual.
- *OpenSimConnector*: esta clase contiene los métodos necesarios para: 1) conectarse al mundo virtual, 2) métodos para escuchar los canales de chat o mensajes directos desde el mundo virtual, y 3) escuchar el progreso de la autentificación.
- *OpenSimObject*: esta clase contiene información específica de un objeto.

Paquete Executor

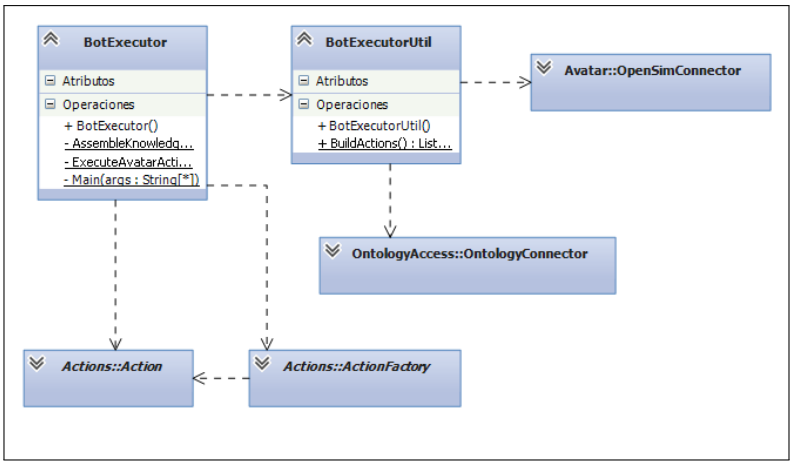

Figura 4.30: Diagrama de Clases del Paquete Executor

En este paquete se encuentran definidas las clases:

- *BotExecutor*: es la clase principal de la aplicación. En esta se construyen las acciones de conocimiento y se ejecutan las acciones del avatar dentro del mundo virtual.
- *BotExecutorUtil*: es una clase utilitaria para la construcción de las acciones de conocimiento.

Paquete OntologyAccess

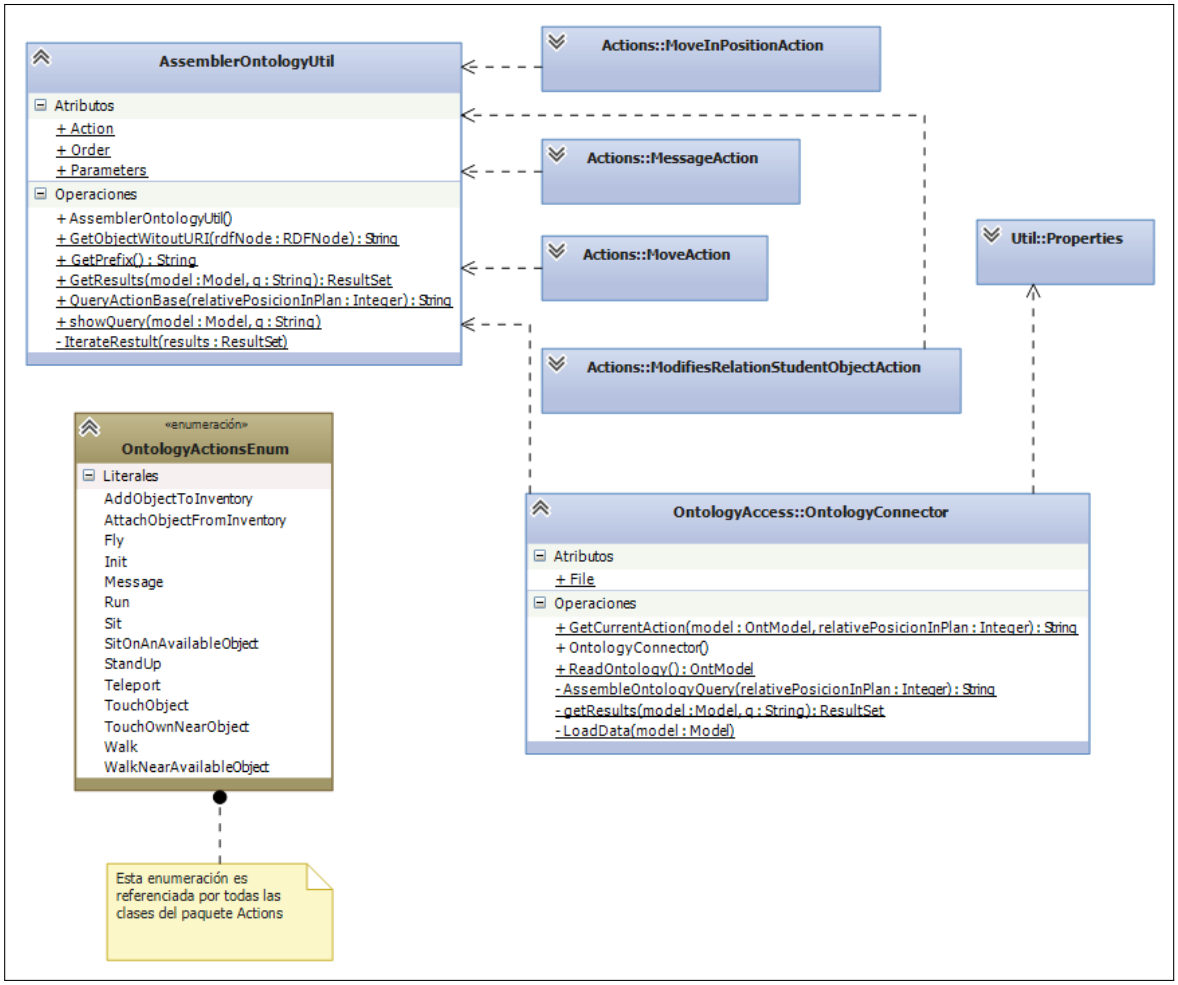

Figura 4.31: Diagrama de Clases del Paquete OntologyAccess

En este paquete se encuentran definidas las clases:

- *AssemblerOntologyUtil*: es una clase utilitaria para la lectura de la ontología y consultas a esta mediante *SPARQL*.
- *OntologyActionsEnum*: es una clase de tipo *enum* que enlista las acciones definidas en la ontología.
- *OntologyConnector*: es la clase que contiene los métodos principales para: 1) leer la ontología, 2) cargar en memoria los datos de la ontología, 3) armar las consultas de la ontología en *SPARQL* y 4) obtener los nombres de las acciones de la ontología.

Paquete Util

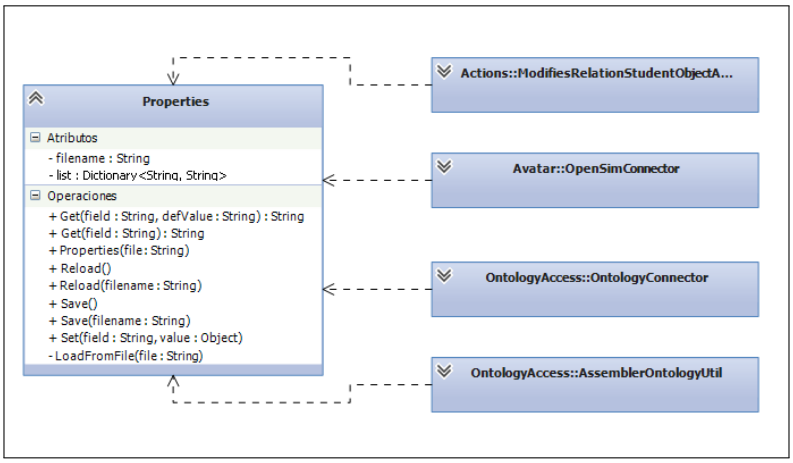

Figura 4.32: Diagrama de Clases del Paquete Util

En este paquete se encuentran definidas las clases:

- *Properties*: esta clase se encarga de leer un archivo de propiedades, donde se encuentras todas la variables del sistema. Tiene un método *Get* que recibe la clave del archivo de propiedades que se requiere obtener.

Paquete Actions

A continuación se muestra el diagrama de clases del paquete *Actions*:

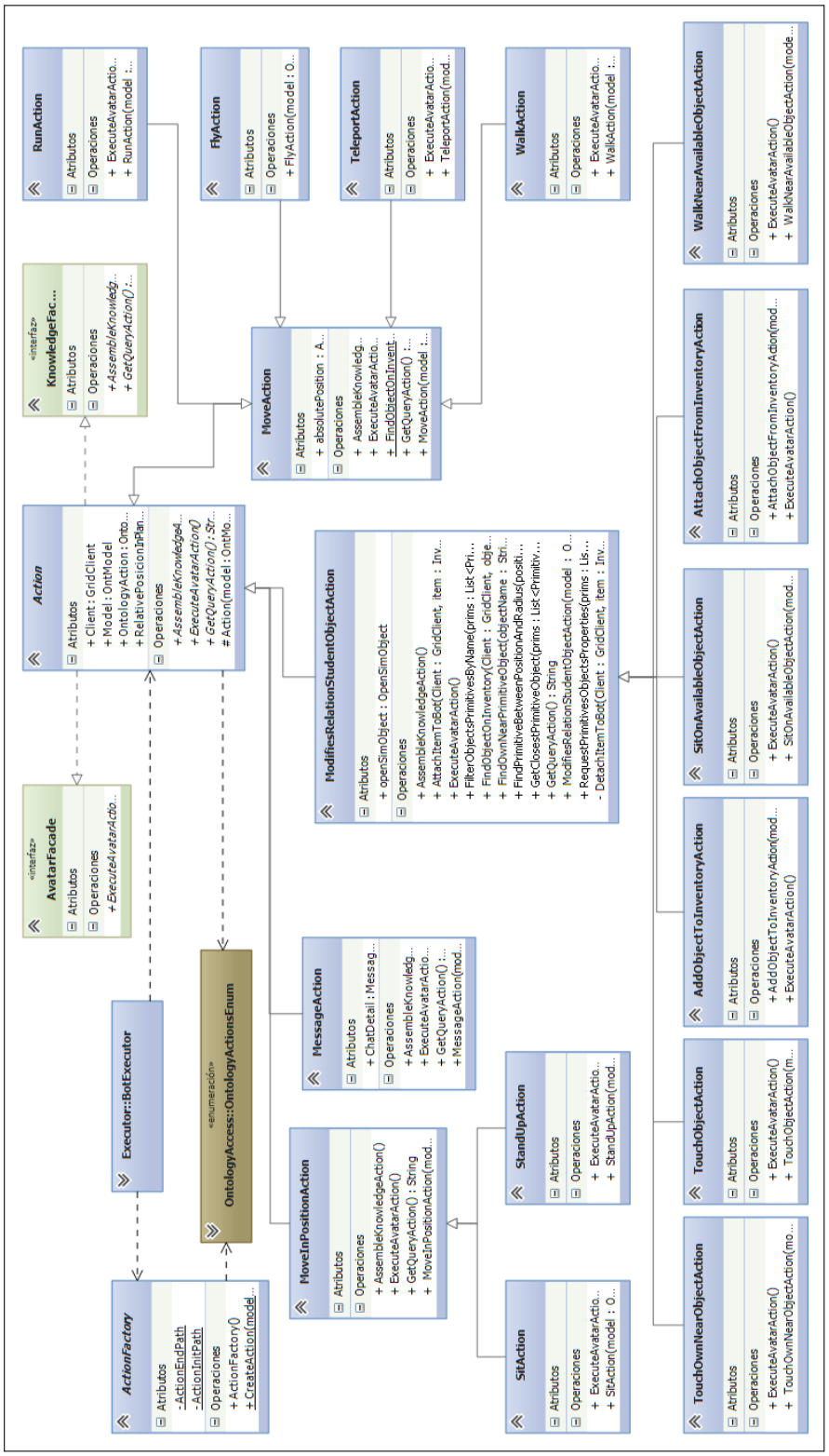

Figura 4.33: Diagrama de Clases del Paquete Actions

En este paquete se tienen las interfaces referentes al conocimiento *KnowledgeFacade* y ejecución de acciones del NPC en el mundo virtual *AvatarFacade*. Además, se encuentran las clases para cada acción que es obtenida leyendo la ontología y luego ejecutada en el mundo virtual. Cada subclase de acción implementa los métodos abstractos declarados en la clase *Action*. La clase que se encarga de crear dinámicamente cada clase de acción es *ActionFactory*.

Las clases de acciones que contienen métodos comunes o genéricos, son agrupados en clases padre, como en el caso de las clases *ModifiesRelationStudentObjectAction*, *MoveAction* y *MoveInPositionAction*.

En este paquete se encuentran definidas las clases:

- *AvatarFacade*: es la interface correspondiente al avatar.
- *KnowledgeFacade* es la interface correspondiente al conocimiento.
- *ModifiesRelationStudentObjectAction*: esta clase implementa la clase abstracta *Action* e implementa los métodos para leer la ontología. Además, es la clase base para las acciones correspondientes a manejo de objetos y contiene métodos para la búsqueda y manipulación de objetos en el mundo virtual. Los métodos comunes que heredan las clases hijos son *GetCurrentAction*, *AssembleKnowledgeAction*, *AttachItemToBot*, *DetachItemToBot*, *FindObjectOnInventory*, *FindOwnNearPrimitiveObject*, *FindPrimitiveBetweenPositionAndRadius*, *RequestPrimitivesObjectsProperties*, *FilterObjectsPrimitivesByName* y *GetClosestPrimitiveObject*. Esta clase es la clase padre de las siguientes clases:
  - *TouchObjectAction*: esta clase extiende la clase *ModifiesRelationStudentObjectAction* e implementa la acción de tocar un objeto específico dentro del mundo virtual.
  - *AddObjectToInventoryAction*: esta clase extiende la clase *ModifiesRelationStudentObject* e implementa la acción añadir un objeto al inventario dentro del mundo virtual.
  - *AttachObjectFromInventoryAction*: esta clase extiende la clase *ModifiesRelationStudentObjectAction* e implementa la acción de añadir un item al inventario del avatar en el mundo virtual.
  - *SitOnAvailableObjectAction*: esta clase extiende la clase *ModifiesRelationStudentObjectAction* e implementa la acción de sentar al avatar en un objeto específico.
  - *WalkNearAvailableObjectAction*: esta clase extiende la clase *ModifiesRelationStudentObjectAction* e implementa la acción de hacer caminar al avatar hasta un objeto específico dentro del mundo virtual.
- *MoveInPositionAction*: esta clase implementa la clase abstracta *Action* e implementa los métodos para leer la ontología. Es la clase base para las acciones de movimiento en la misma posición del avatar. Los métodos comunes que heredan las clases hijos son *GetCurrentAction* y *AssembleKnowledgeAction*. Es la clase padre de:
  - *SitAction*: esta clase extiende la clase *MoveInPositionAction* e implementa la acción de sentar al avatar en la posición en donde se encuentre dentro del mundo virtual.
  - *StandUpAction*: esta clase extiende la clase *MoveInPositionAction* e implementa la acción hacer parar al avatar en la posición en donde se encuentre dentro del mundo virtual.

- *MoveAction*: esta clase implementa la clase abstracta *Action* e implementa los métodos para leer la ontología. Es la clase base para las acciones de movimiento del avatar. Los métodos comunes que heredan las clases hijos son *GetCurrentAction*, *AssembleKnowledgeAction* y *FindObjectOnInventory*. Es la clase padre de las siguientes clases:
  - *FlyAction*: esta clase extiende de la clase *MoveAction* y es una clase base para las acciones de movimiento del avatar en el mundo virtual.
  - *WalkAction*: esta clase extiende la clase *MoveAction* e implementa la acción de hacer caminar al avatar a una posición específica dentro del mundo virtual.
  - *RunAction*: esta clase extiende la clase *MoveAction* e implementa la acción de cambiar el estado a corriendo del avatar dentro del mundo virtual.
  - *TeleportAction*: esta clase extiende la clase *MoveAction* e implementa la acción de teletransportar al avatar a una posición específica dentro del mundo virtual.
- *MessageAction*: esta clase implementa la clase abstracta *Action* e implementa los métodos para leer la ontología y ejecutar la acción de enviar un mensaje al mundo virtual.

4.2.2.2. Diagramas de Secuencia

Secuencia BotExecutor.Main

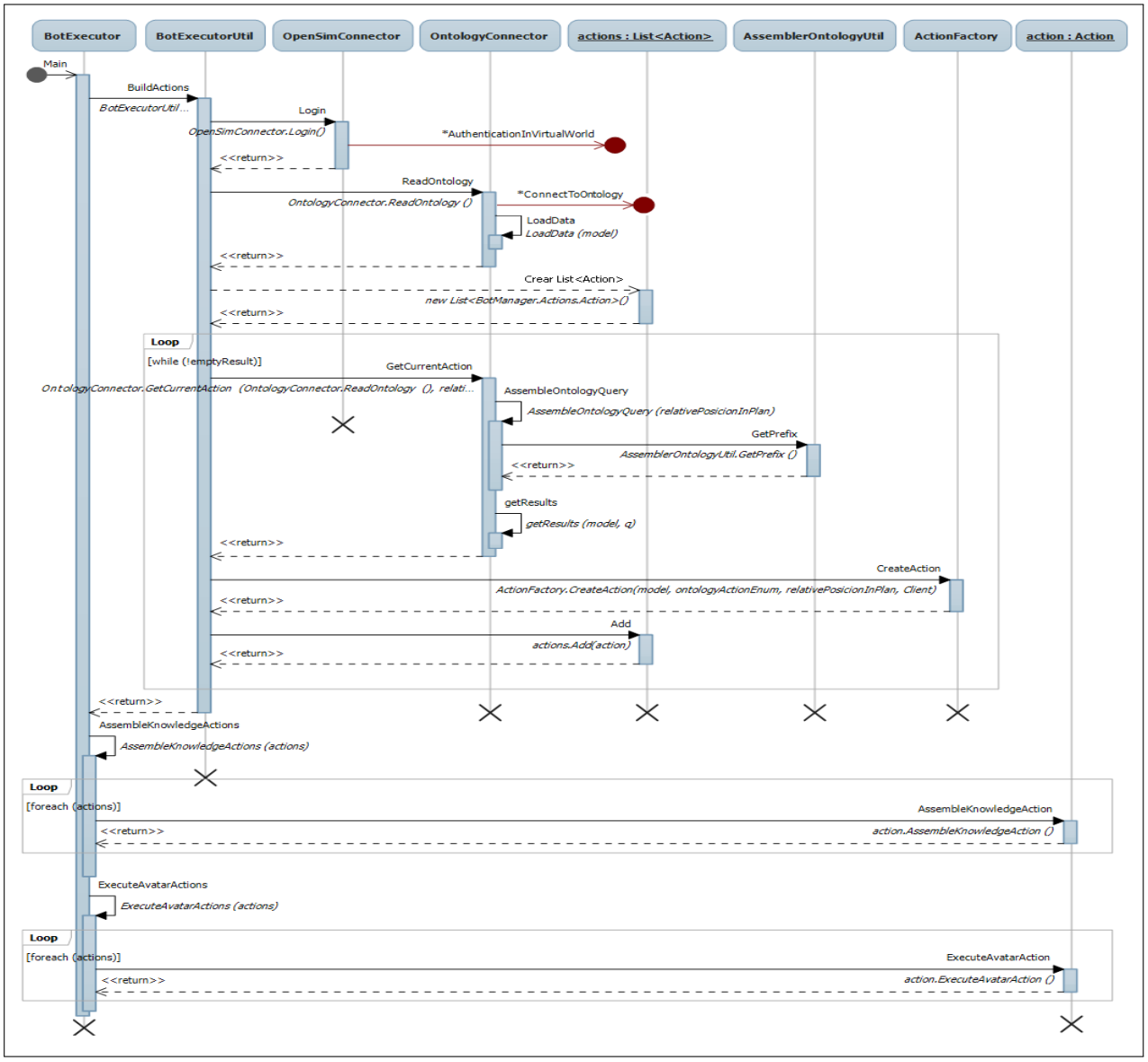

Figura 4.34: Diagrama de Secuencia del método BotExecutor.Main

La figura 4.34 muestra el diagrama de secuencia del método *Main* de la clase *BotExecutor*. Este método es el principal y por defecto es invocado al ejecutar la aplicación BotManager.

Inicialmente se hace la invocación al método *BotExecutorUtil.BuildActions* ([Nombre de la clase].[Nombre del método]), el cual invoca al método *OpenSimConnector.Login()* que se conecta al mundo virtual a través de las APIs de *OpenSimulator*, en el diagrama se denota con la llamada

**AuthenticationInVirtualWorld*, cuyo pseudocódigo se detalla a continuación:

Data:
Result: objeto *Client* del tipo *GridClient* que contiene los métodos para controlar el NPC e interacción con este dentro del mundo virtual
lee un archivo de propiedades a través de la clase *Properties*, donde se almacenan las variables: *openSimPath* que es la ruta del servidor de OpenSimulator, *OpenSimPort* el puerto del servidor, *firstName* nombre de usuario, *lastName* apellido del usuario , *password* clave y *location* que es la región del mundo virtual; Definir los valores de configuración del servidor de OpenSimulator *Client.Settings.LOGIN_SERVER = openSimPath + ":" + openSimPort*;
Definir los métodos escuchadores de eventos de mensajes instantáneos del servidor *Client.Self.IM += new EventHandler<InstantMessageEventArgs>(SelfIM)*;
Definir los métodos escuchadores de eventos de mensajes chat generales del servidor *Client.Self.ChatFromSimulator += new EventHandler<ChatEventArgs>(Self_ChatFromSimulator)*;
if *El firstName, lastName, password y location son correctos* **then**
    retornar *Client* ;
else
    retornar null;
end

Algorithm 1: Método OpenSimConnector.Login y llamadas *AuthenticationInVirtualWorld

Si el objeto retornado es nulo la aplicación termina, de no ser así, el siguiente paso es acceder a la ontología con el método *OntologyConnector.ReadOntology* que devuelve el objeto *model* del tipo *OntModel*, que se conecta a la ontología del estudiante a través de las APIs de Jena, en diagrama se denota con la llamada **ConnectToOntology*, que se detalla a continuación:

Data:
Result: objeto del tipo *OntModel* que contiene los métodos para acceder a la ontología
lee un archivo de propiedades a través de la clase *Properties*, donde se almacenan las variables: *source* es la ruta física donde se encuentran los archivos OWL de la ontología, *knowledgeObjectEntry* es el url de la ontología el estudiante, *knowledgeObjectFilename* el nombre del archivo de la ontología del estudiante, *botManagerEntry* es el url de la ontología BotManager que extiende de la ontología del estudiante, *botManagerFilename* es el nombre del archivo de la ontología BotManager;
Creación de objeto *model* de la clase *OntModel* tras la invocación del método *ModelFactory.createOntologyModel*;
Se añaden las entradas de la ontología con el método *model.getDocumentManager().addAltEntry*, que recibe como parámetros *knowledgeObjectEntry*, *source* y *knowledgeObjectFilename*;
retornar model;

Algorithm 2: Método OntologyConnector.ReadOntology y llamadas *ConnectToOntology

Consecuentemente en la misma clase *OntologyConnector* se ejecuta el método *LoadData*, que lee la ontología de instancias del BotManager tras invocar al método *FileManager.get().readModel*, que recibe el *model*, *source* más el nombre de la instancia de la ontología (leída del archivo de propiedades). Esto retorna el *model*. El siguiente paso es crear un lista de objetos del tipo *Action*

almacenado en variable *actions*.

Luego se recorre hasta que la acción leída de la ontología sea nula. En la iteración dentro de *BotExecutorUtil.BuildActions* se llama a *OntologyConnector.GetCurrentAction* que recibe los parámetros *model* y *relativePositionInPlan*. *OntologyConnector.GetCurrentAction* invoca al método propio *AssembleOntologyQuery* que recibe *relativePositionInPlan*, el cual devuelve una consulta en lenguaje SPARQL para obtener la ontología actual que se esta iterando, el resultado devuelto almacenado en la variable *q*, es la respuesta de invocar a *AssemblerOntologyUtil.GetPrefix* más la consulta base de la consulta SPARQL para obtener la acción actual. El siguiente método invocado en *OntologyConnector.GetCurrentAction* es *GetResults* que recibe *model* y *q* que retorna un objeto *ResultSet*, el cual se recorre y se obtiene el nombre de la acción de la ontología en URL de RDF, que es el resultado devuelto del método pero sin el URL, al aplicar el método *String.Substring*.

Luego continuando en el método *BotExecutorUtil.BuildActions*, se crea la acción en el objeto *action* del tipo *Action* a través del método *ActionFactory.CreateAction* el cual recibe los parámetros *model*, *ontologyActionEnum*, *relativePositionInPlan* y *Client*. Usando reflexión convierte mediante el nombre de la ontología, recibido en el parámetro *ontologyActionEnum* en la clase de acción especifica para dicha acción y crea un objeto *action* de ese tipo. Finalmente, esta *action* se agrega a la lista de acciones *actions*.

El siguiente paso en *BotExecutor.Main* es ejecutar los métodos *AssembleKnowledgeActions* y *ExecuteAvatarActions* para la lista de acciones *actions*. Eso implica que para cada acción de la lista, se ejecutan los métodos *action.AssembleKnowledgeAction* y *action.ExecuteAvatarAction* respectivamente. El primero de estos dos métodos accede a la ontología para obtener los argumentos de cada acción y el segundo método manda al NPC que ejecute la acción que se acaba de leer de la ontología.

Secuencia OntologyConnector.GetCurrentAction

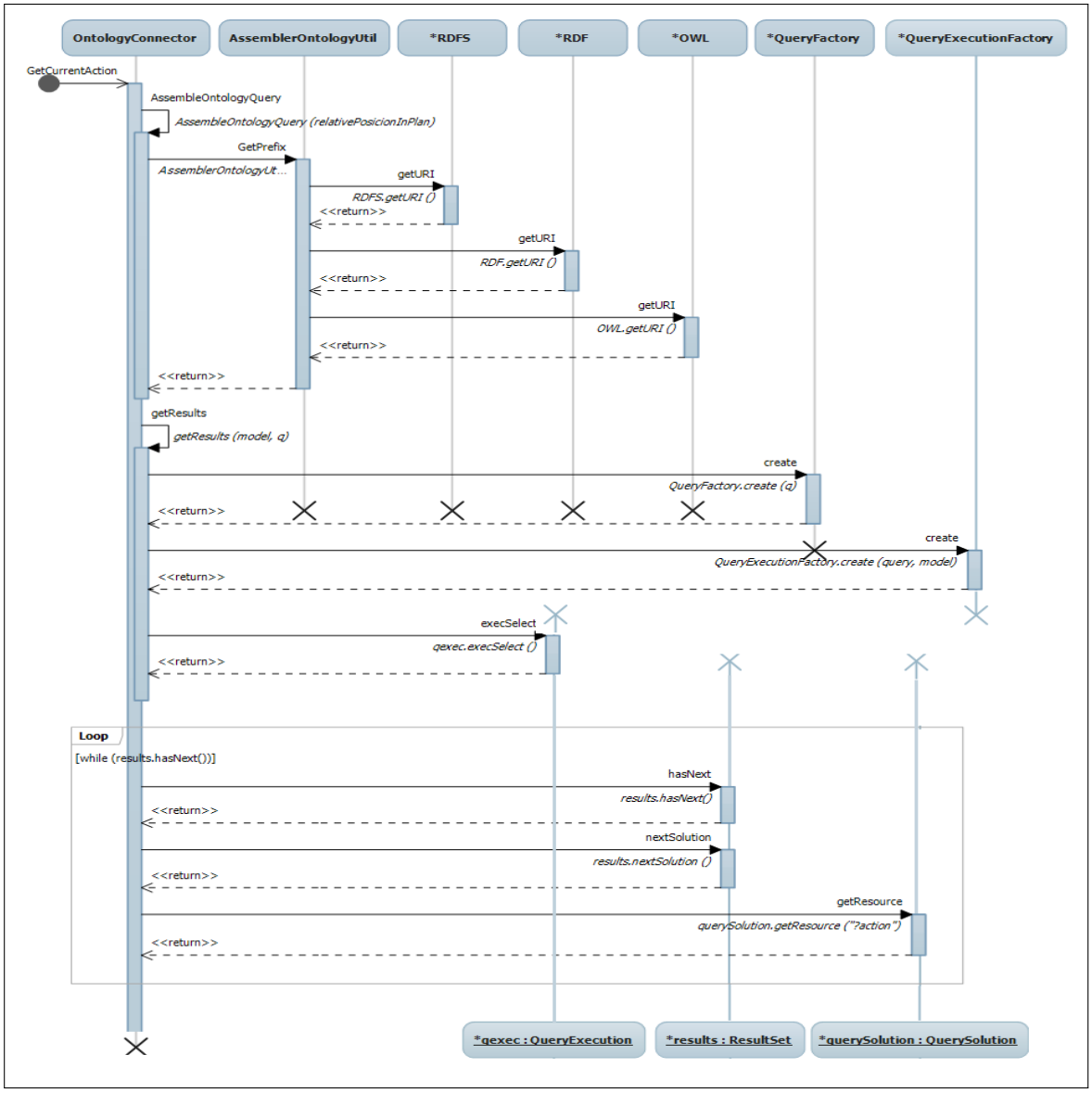

Figura 4.35: Diagrama de Secuencia del método OntologyConnector.GetCurrentAction

La figura 4.34 muestra el diagrama de secuencia del método *OntologyConector.GetCurrentAction* y su interacción con clases de BotManager y la interacción con las clases de las librerías externas del API de Jena precedidas con (*) al inicio del nombre de la clase. *OntologyConector.GetCurrentAction* recibe como parámetros el objeto *model* del tipo *OntModel* y la posición de la acción dentro del plan en la ontología *relativePositionInPlan*. Primeramente se llama al método de la misma clase *AssembleOntology* que recibe *relativePositionInPlan*, este método realiza la invocación a *AssemblerOntologyUtil.GetPrefix* (detallado en 4.34) y las siguientes invocaciones a métodos de clases de las librerías de Jena:

- **RDF.getURI* obtiene el URL del esquema RDF.
- **OWL.getURI* obtiene el URI del esquema OWL.

El siguiente paso, es la llamada al método de la misma clase *getResults* (detallado en 4.34). Como siguiente paso son las siguientes invocaciones a métodos de clases de las librerías de Jena:

- **QueryFactory.create* crea el objeto *query* del tipo **Query*.
- **QueryExecutionFactory.create* crea el objeto *qexec* del tipo **QueryExecution*.
- **qexec.QueryExecution* que devuelve el objeto *results* del tipo **ResultSet*

Luego se recorre *results* y se hacen las siguientes invocaciones a métodos de clases de librerías de Jena:

- *results.hasNext* hacer iterar mientras *results* tenga elementos.
- *results.nextSolution* obtiene el objeto *querySolution* del tipo *querySolution*.
- *querySolution.getResource* obtiene el nombre de la acción y lo almacena en la variable *actionString*.

Si *actionString* es nulo deja de iterar. Y finalmente *OntologyConector.GetCurrentAction* retorna *actionString*.

Capítulo 5

Pruebas

En esta sección se desarrollará la practica de "preparación de una taza de café" (Fernández-Avilés Pedraza, 2013), que pertenece a la primera práctica de un Laboratorio Virtual de Biotecnología (Riofrío Luzcando, 2012) desarrollado sobre OpenSimulator. Esta práctica fue creada, a modo de tutorial previo a la práctica de biotecnología propiamente dicha, para que los usuarios se familiaricen con el mundo virtual y la forma de interactuar con los objetos del laboratorio. La ejecución de esta práctica la realizará el BotManager. Para esto es necesario crear un archivo de instancias de la ontología *Knowledge_Object_Bot_Manager* llamado *Knowledge_Object_Bot_Manager_Instances*. Donde se detallan paso a paso las acciones que tiene que realizar el NPC en el laboratorio virtual.

5.1. Listado de las Acciones del Plan de Prueba

1. Teletransportar al NPC a la posición inicial.
2. Caminar hasta en frente de la puerta de entrada al laboratorio.
3. Tocar la puerta de entrada para abrirla.
4. Caminar hasta en frente de la máquina de selección de prácticas.
5. Tocar la pared de prácticas.
6. Seleccionar la práctica número 1, que corresponde al tutorial "preparación de una taza de café".
7. Caminar hasta en frente de la mesa con tazas.
8. Tocar la mesa con tazas (se entrega una taza al inventario del NPC).
9. Tomar taza desde el inventario del NPC.
10. Caminar hasta en frente de la máquina de café.
11. Tocar máquina de café (para seleccionar los ingredientes).

12. Añadir café a la taza.
13. Añadir leche a la taza.
14. Añadir azúcar a la taza.
15. Caminar hasta cerca de una mesa de café disponible.
16. Soltar la taza en la mesa.
17. Sentarse en la silla más cercana a donde se soltó la taza.
18. Tocar la taza de café del NPC (para tomar el café).

5.2. Definición de las Instancias para el Plan de Prueba

Teletransportar al NPC a la posición inicial

Clase	**Teleport**.
Propiedad	relativePosicionInPlan: 1.
Propiedad	associatedAction->Teleport->destinationPos->Absolute_Position: [coordinateX: 131, coordinateY: 50, coordinateZ: 21].

Tabla 5.1: Clase Teleport

Caminar hasta en frente de la puerta de entrada al laboratorio

Clase	**Walk**.
Propiedad	relativePosicionInPlan: 2.
Propiedad	associatedAction->Walk->destinationPos->Absolute_Position: [coordinateX: 131, coordinateY: 54, coordinateZ: 23].

Tabla 5.2: Clase Walk

Tocar la puerta de entrada

Clase	**Touch_Object**.
Propiedad	relativePosicionInPlan: 3.
Propiedad	associatedAction->Touch_Object->isAppliedToObjects->Object: [idObject: "e220c9c9-40d6-4fe3-8097-d857663368bf", descriptorObject: "PuertaEntrada"].

Tabla 5.3: Clase Touch_Object

Caminar hasta en frente de la máquina de prácticas

Clase	**Walk**.
Propiedad	relativePosicionInPlan: 4.
Propiedad	associatedAction->Walk->destinationPos->Absolute_Position: [coordinateX: 135, coordinateY: 78, coordinateZ: 23].

Tabla 5.4: Clase Walk

Tocar la pared de prácticas

Clase	**Touch_Object**.
Propiedad	relativePosicionInPlan: 5.
Propiedad	associatedAction->Touch_Object->isAppliedToObjects->Object: [idObject: "477de945-c4ea-4ca9-bd5a-ac34f660906e", descriptorObject: "ParedPractica"].

Tabla 5.5: Clase Touch_Object

Seleccionar la práctica número 1, que corresponde al tutorial "preparación de una taza de café"

Clase	**Message**.
Propiedad	relativePosicionInPlan: 6.
Propiedad	associatedAction->Message->chatToSimulator->Message_Detail [channel: 4, type:"Normal", message: "1"].

Tabla 5.6: Clase Message

Caminar hasta en frente de la mesa con tazas

Clase	**Walk**.
Propiedad	relativePosicionInPlan: 7.
Propiedad	associatedAction->Walk->destinationPos->Absolute_Position: [coordinateX: 122, coordinateY: 77, coordinateZ: 22].

Tabla 5.7: Clase Walk

Tocar la mesa con tazas

Clase	**Touch_Object**.
Propiedad	relativePosicionInPlan: 8.
Propiedad	associatedAction->Touch_Object->isAppliedToObjects->Object: [idObject: “69d1e4bf-0766-4eed-b16f-3f5b382f3c1a”, descriptorObject: “SeleccionTaza”].

Tabla 5.8: Clase Touch_Object

Tomar taza desde el inventario del NPC

Clase	**Attach_Object_From_Inventory**.
Propiedad	relativePosicionInPlan: 9.
Propiedad	associatedAction->Attach_Object_From_Inventory->Cup: [name: “Taza de cafe”].

Tabla 5.9: Clase Attach_Object_From_Inventory

Caminar hasta en frente de la máquina de café

Clase	**Walk**.
Propiedad	relativePosicionInPlan: 10.
Propiedad	associatedAction->Walk->destinationPos->Absolute_Position: [coordinateX: 123:, coordinateY: 78, coordinateZ: 23].

Tabla 5.10: Clase Walk

Tocar máquina de café

Clase	**Touch_Object**.
Propiedad	relativePosicionInPlan: 11.
Propiedad	associatedAction->Touch_Object->isAppliedToObjects->Simple_Object: [idObject: “27d2872d-3a2e-47cb-90e9-40d228e920d3”, name: “Maquina Cafe”].

Tabla 5.11: Clase Touch_Object

Añadir café a la taza

Clase	**Message**.
Propiedad	relativePosicionInPlan: 12.
Propiedad	associatedAction->Message->chatToSimulator->Message_Detail [channel:30, type:“Normal”, message:“1”].

Tabla 5.12: Clase Message

Añadir leche a la taza

Clase	**Message**.
Propiedad	relativePosicionInPlan: 13.
Propiedad	associatedAction->Message->chatToSimulator->Message_Detail [channel:30, type:“Normal”, message:“2”].

Tabla 5.13: Clase Message

Añadir azúcar a la taza

Clase	**Message**.
Propiedad	relativePosicionInPlan: 14.
Propiedad	associatedAction->Message->chatToSimulator->Message_Detail [channel:30, type:“Normal”, message:“3”].

Tabla 5.14: Clase Message

Caminar hasta cerca de una mesa de café disponible

Clase	**Walk_Near_Available_Object**.
Propiedad	relativePosicionInPlan: 15.
Propiedad	associatedAction->Walk_Near_Available_Object->isAppliedToObjects->Simple_Object: [name: “MesaCafe”].

Tabla 5.15: Clase Walk_Near_Available_Object

Soltar la taza en la mesa

Clase	**Message**.
Propiedad	relativePosicionInPlan: 16.
Propiedad	associatedAction->Message->chatToSimulator->Message_Detail [channel:507, type:“Normal”, message:“dropTaza”]..

Tabla 5.16: Clase Message

Sentarse en la silla más cercana a donde se soltó la taza

Clase	**Sit_On_An_Available_Object**.
Propiedad	relativePosicionInPlan: 17.
Propiedad	associatedAction->Sit_On_An_Available_Object->isAppliedToObjects->Simple_Object: [name: “sillaCafe”].

Tabla 5.17: Clase Sit_On_An_Available_Object

Tocar la taza de café del NPC

Clase	**Touch_Own_Near_Object**.
Propiedad	relativePosicionInPlan: 18.
Propiedad	associatedAction->Touch_Own_Near_Object->isAppliedToObjects->Simple_Object: [name: “Taza de cafe”].

Tabla 5.18: Clase Touch_Own_Near_Object

5.3. Ejecución del Plan de Prueba

Teletransportar al NPC a la posición inicial

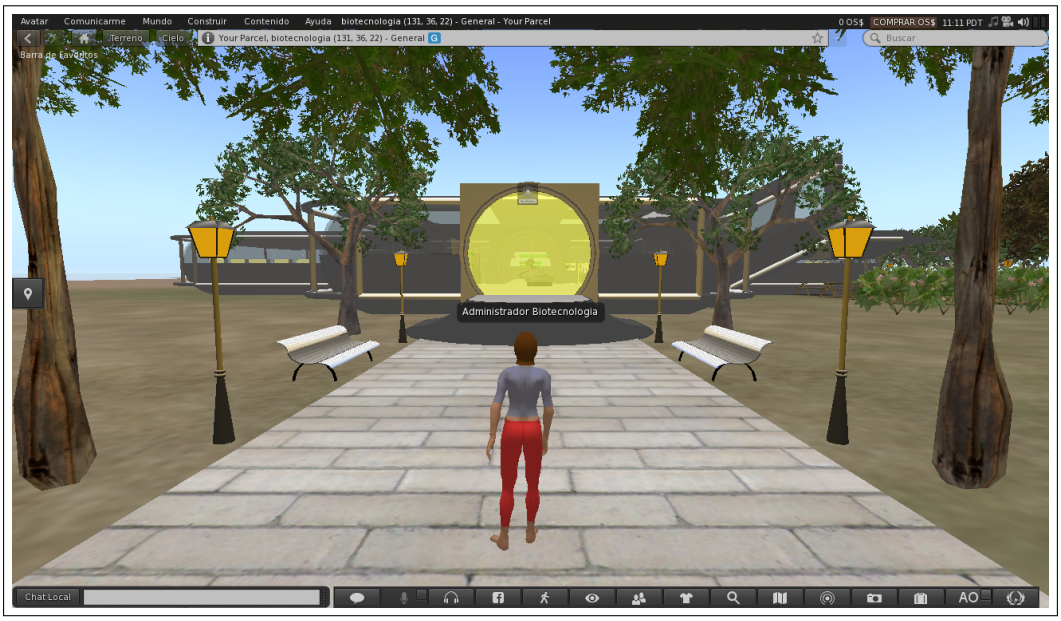

Figura 5.1: Teletransportar al NPC a la posición inicial

Caminar hasta en frente de la puerta de entrada al laboratorio

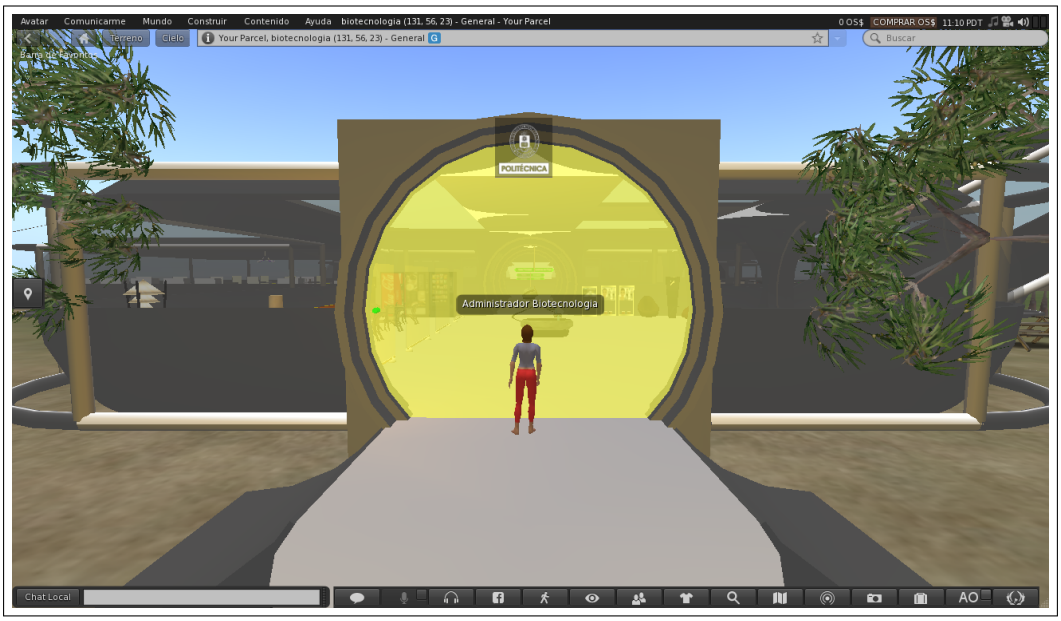

Figura 5.2: Caminar hasta en frente de la puerta de entrada al laboratorio

Tocar la puerta de entrada

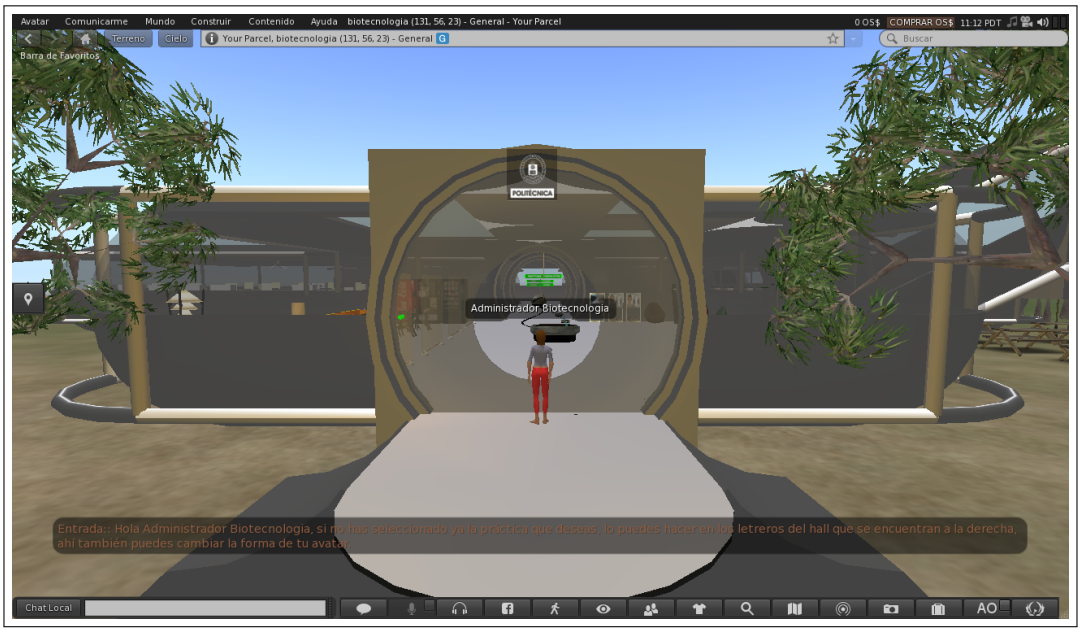

Figura 5.3: Tocar la puerta de entrada

Caminar hasta en frente de la máquina de prácticas

Figura 5.4: Caminar hasta en frente de la máquina de prácticas

Tocar la máquina de prácticas

Figura 5.5: Tocar la máquina de prácticas

Seleccionar la práctica número 1, que corresponde al tutorial "preparación de una taza de café"

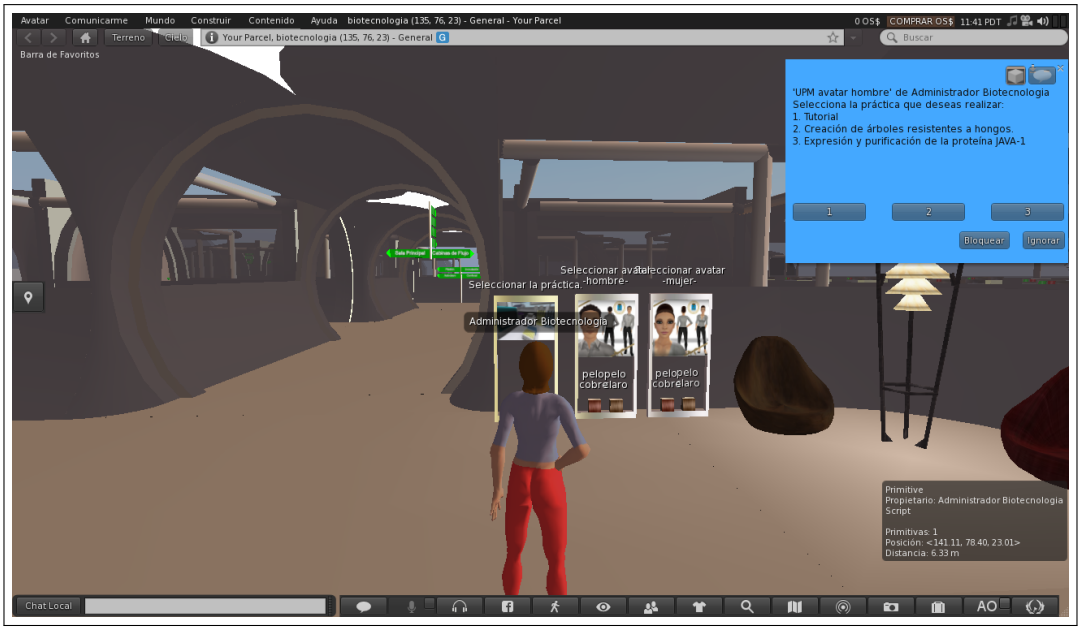

Figura 5.6: Seleccionar la práctica número 1, que corresponde al tutorial "preparación de una taza de café"

Caminar hasta en frente de la mesa de con tazas

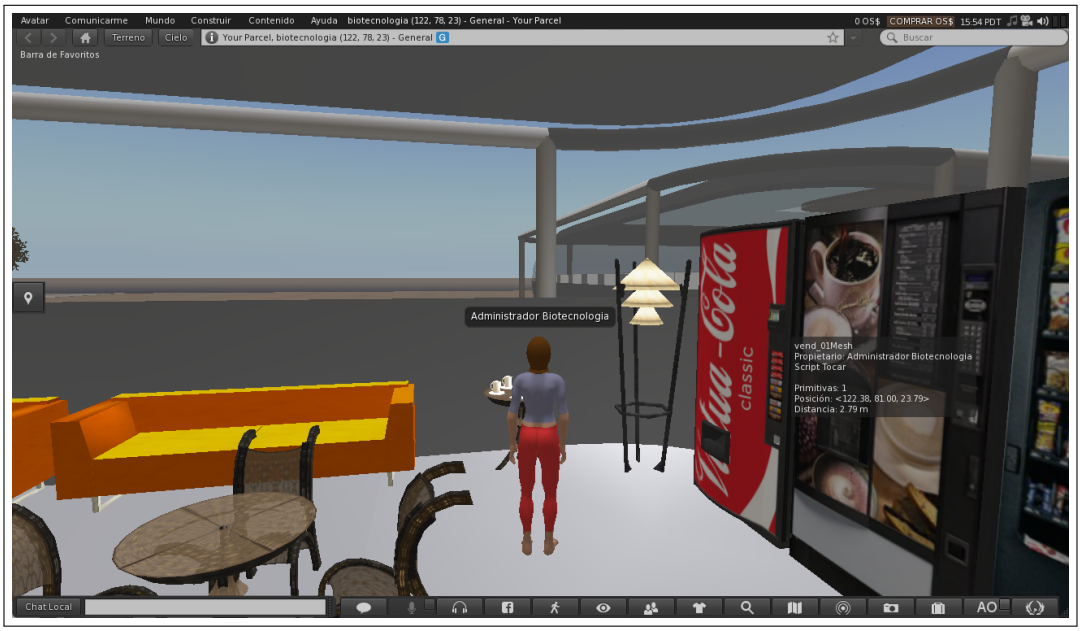

Figura 5.7: Caminar hasta en frente de la mesa de con tazas

Tocar la mesa con tazas

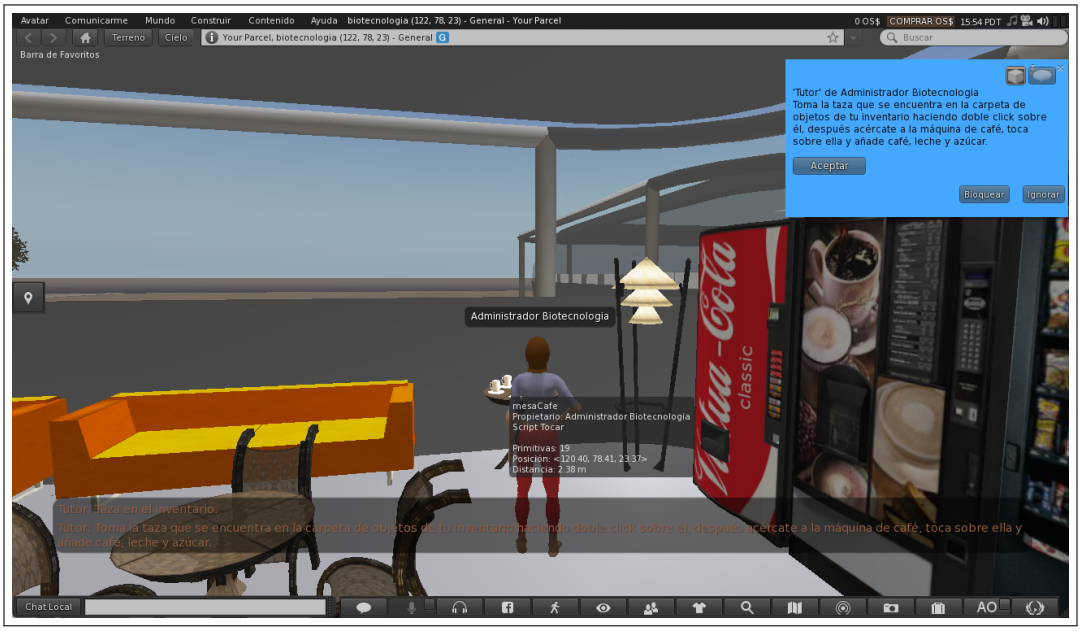

Figura 5.8: Tocar la mesa con tazas

Tomar taza desde el inventario del NPC

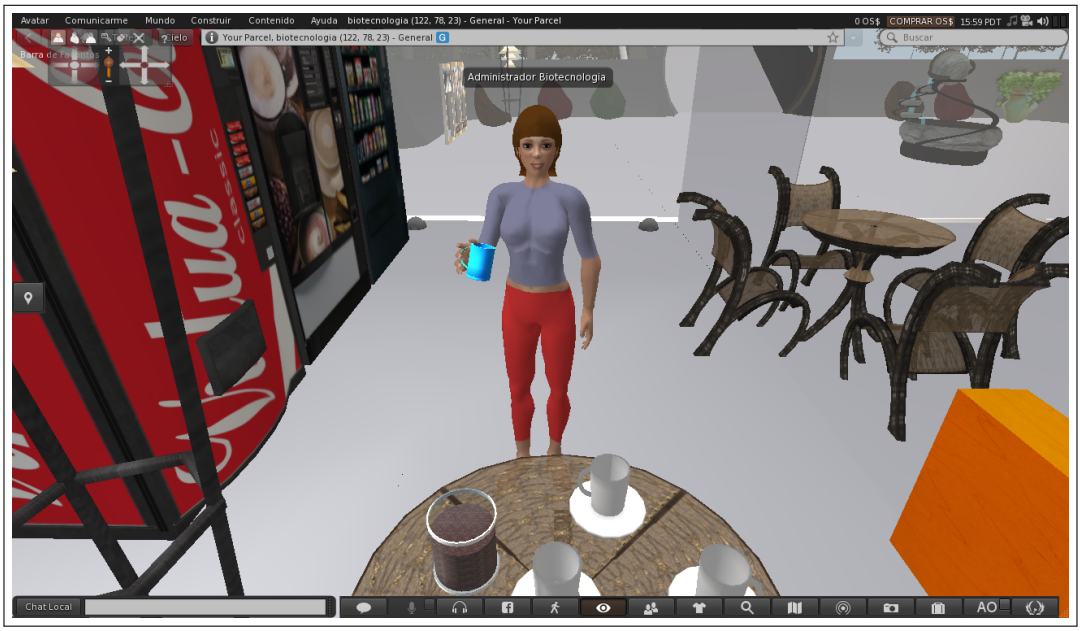

Figura 5.9: Tomar taza desde el inventario del NPC

Caminar hasta en frente de la máquina de café

Figura 5.10: Caminar hasta en frente de la máquina de café

Tocar máquina de café

Figura 5.11: Tocar máquina de café

Añadir café a la taza

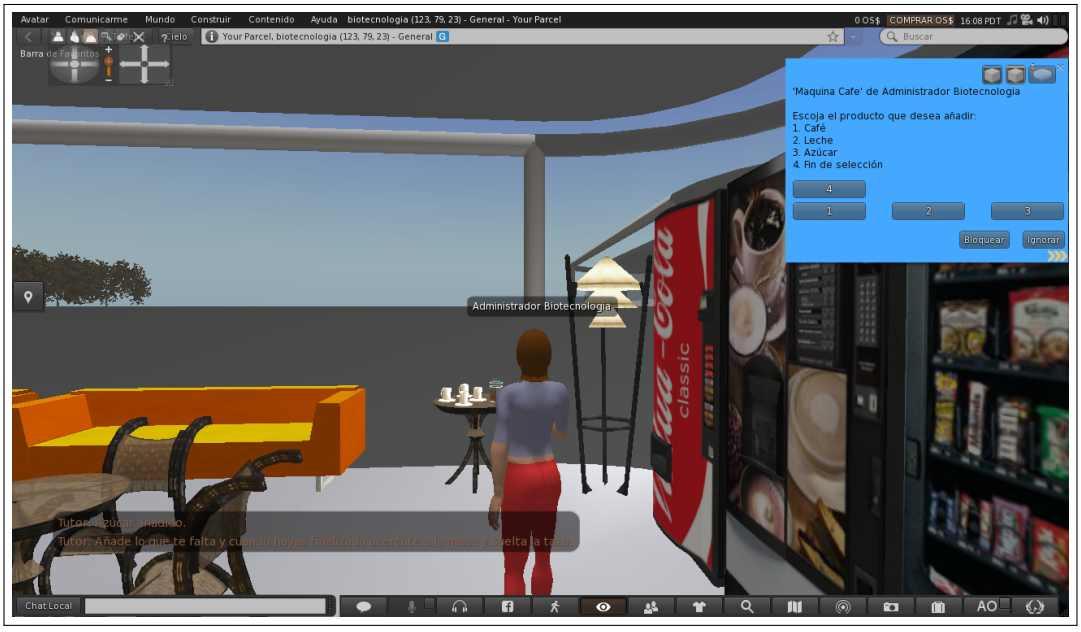

Figura 5.12: Añadir café a la taza

Añadir leche a la taza

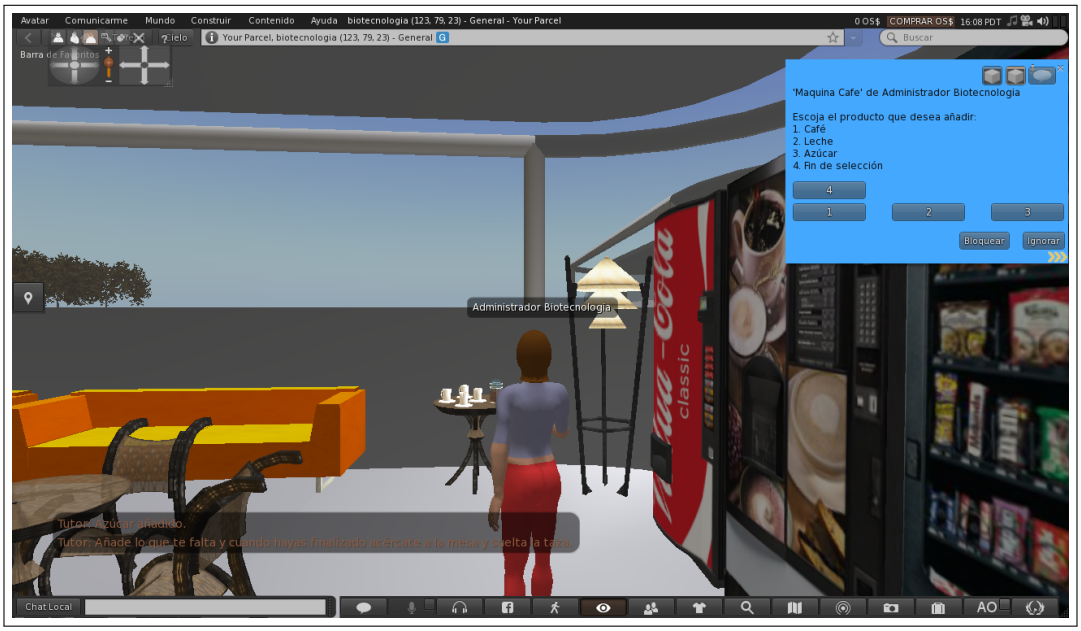

Figura 5.13: Añadir leche a la taza

Añadir azúcar a la taza

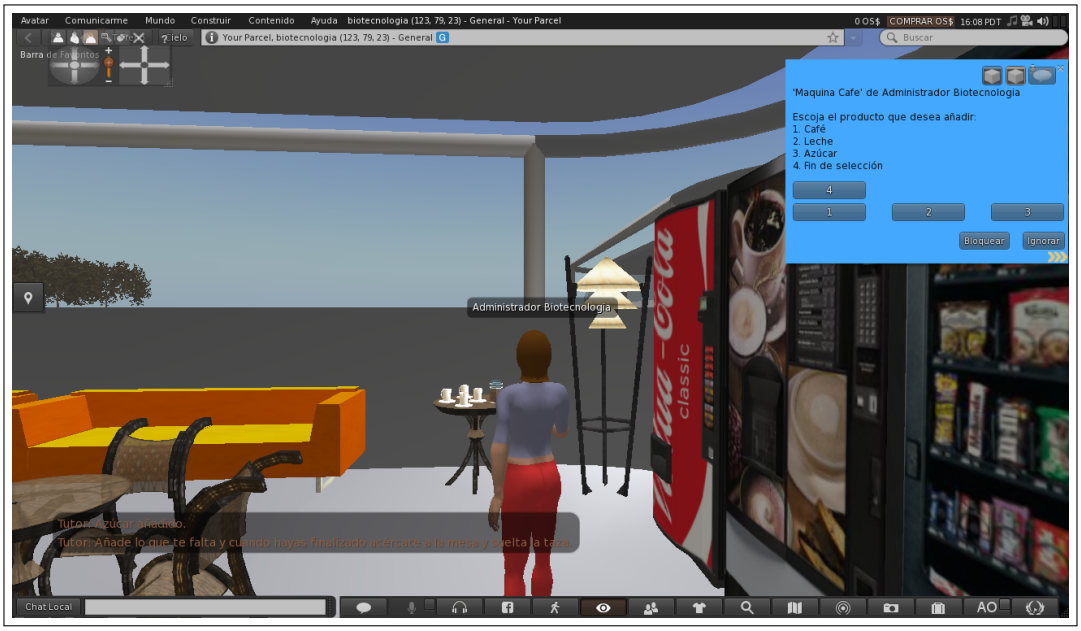

Figura 5.14: Añadir azúcar a la taza

Caminar hasta cerca de una mesa de café disponible

Figura 5.15: Caminar hasta cerca de una mesa de café disponible

Soltar la taza en la mesa

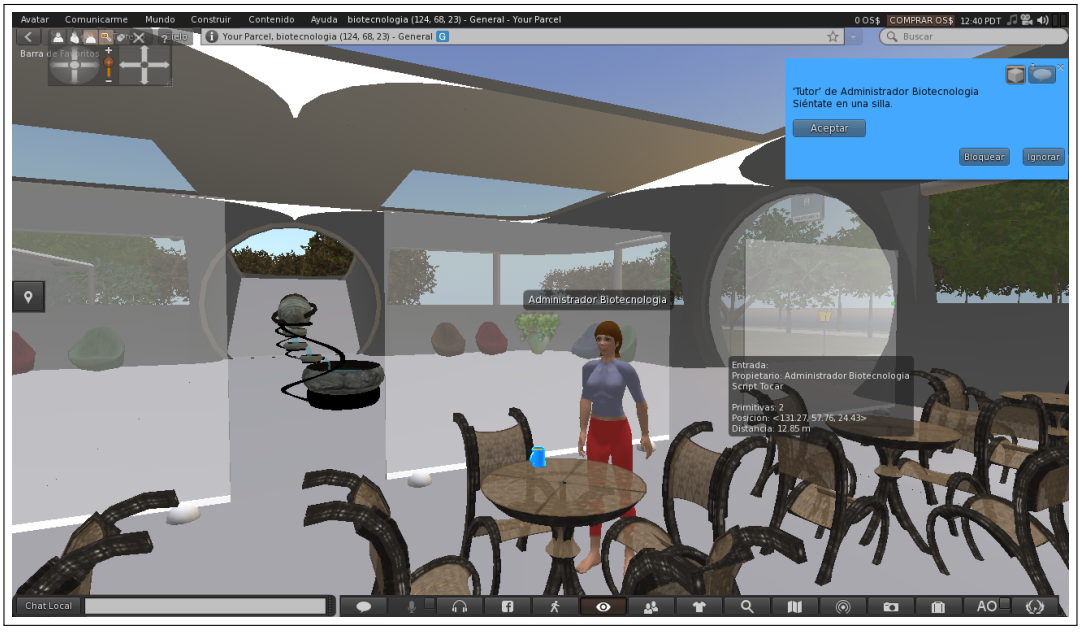

Figura 5.16: Soltar la taza en la mesa

Sentarse en la silla más cercana a donde se soltó la taza

Figura 5.17: Sentarse en la silla más cercana a donde se soltó la taza

Tocar la taza de café del NPC

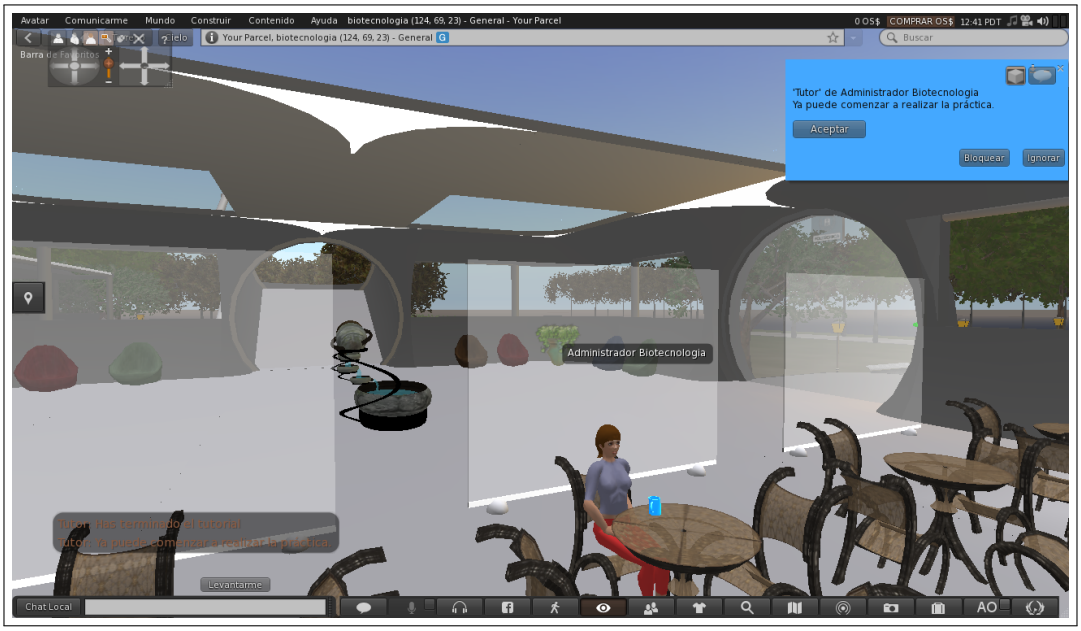

Figura 5.18: Tocar la taza de café

Capítulo 6

Conclusiones

En este trabajo se implementó una aplicación para controlar un NPC dentro de un mundo virtual, en donde el conocimiento sobre las acciones a realizar se encuentra en una ontología. Las herramientas seleccionadas para la construcción de la aplicación BotManager fueron la librería *LibOpenMetaverse* para la interacción con el mundo virtual, la librería *Jena* para el acceso a la ontología, y el lenguaje de programación *C#*. La construcción de la arquitectura de la aplicación y la integración con estas herramientas no presentó especiales dificultades. En cuanto a la documentación con respecto a *Jena* y *C#* cabe señalar que es abundante y se puede encontrar en muchos sitios. Por otra parte, en cuanto a la documentación y ejemplos con *LibOpenMetaverse* cabe decir que es muy limitada, si bien existe documentación de la API, parte de la información sobre los detalles de las clases y los métodos brillaba por su ausencia. Otra limitación de *LibOpenMetaverse* es que ciertas funciones que existen dentro de *OpenSimulator* en el lenguaje *LSL*, no estaban implementadas en *LibOpenMetaverse* como, por ejemplo, la de "soltar un objeto".

Cabe mencionar también que al poco tiempo de finalizar el desarrollo del BotManager, el sitio del proyecto *LibOpenMetaverse* salió a la venta y fue comprado por una empresa que se dedica al diseño Web. Ahora bien, el proyecto puede seguir siendo desarrollado ya que se puede descargar del repositorio de proyectos libres *GitHub*[1]. Hasta la fecha no hay ningún grupo que lidere el desarrollo de *LibOpenMetaverse*, lo que podría implicar que deje de ser utilizada, ya que no hay quien mantenga *LibOpenMetaverse* para las nuevas funcionalidades que sin duda aparecerán en las nuevas versiones de *OpenSimulator*.

El BotManager es una aplicación multipropósito que puede ser utilizada tanto para mandar un listado de acciones a un NPC en un mundo vacío, como complemento para mundos virtuales de enseñanza, capacitación o simulación. Basándose en la ontología *Knowledge_Object_Bot_Manager* que extiende la ontología del estudiante de Julia (Párraga, 2011), y teniendo definido en un archivo de propiedades el nombre de la ontología base, el nombre del fichero de las instancias, la IP del servidor de OpenSimulator, y el usuario y clave del NPC, el BotManager puede ser empleado dentro de cualquier mundo virtual. Ahora bien, en el caso de que la ontología donde

[1] http://www.github.com

se encuentren las acciones a realizar en el mundo virtual no tenga la estructura de la ontología *Knowledge_Object_Bot_Manager*, se deben modificar las clases del paquete *OntologyAccess* para su incorporación.

El BotManager ha sido probado con la práctica "preparación de una taza de café" (Fernández-Avilés Pedraza, 2013), que sirve de iniciación dentro de un Laboratorio virtual de Biotecnología (Riofrío Luzcando, 2012). Esta es una práctica básica en la que el NPC realiza la secuencia de la elaboración de una taza de café paso a paso. En una práctica más compleja, donde el NPC deba comportarse de diferente forma dependiendo de lo que hagan los avatares a su alrededor, y por tanto se requiera manejar información sobre el estado del mundo virtual, se deben incorporar al BotManager las librerías necesarias para obtener dicha información.

Capítulo 7

Trabajo Futuro

En el BotManager se podría crear un módulo para la creación de objetos en tiempo real o escenarios de un mundo virtual dinámicamente usando las librerías de *LibOpenMetaverse*. De esta forma se podría tener toda la información de un mundo virtual definida completamente en una ontología. Otro caso sería el de un NPC que al ejecutar una acción requiera crear o destruir objetos, como por ejemplo, si el NPC parte en dos una tabla con un serrucho. Para simular esta acción hay que eliminar la tabla actual y crear dos objetos "tabla", que representan a la tabla inicial partida en dos. Toda esta información sobre el efecto de la acción "cortar la tabla" estaría definida en la ontología.

Otra adaptación que podría realizarse en el BotManager consistiría en poder controlar múltiples NPCs. Para esto podría implementarse el uso de hilos de ejecución, donde cada hilo controlaría un NPC. Esto podría ser utilizado en simulacros virtuales, como en el caso de una emergencia, donde se requiera la participación ciudadana, y cada persona que maneja un avatar, esté acompañada por un NPC que lo guíe, de ser necesario, por el camino correcto en el simulacro virtual.

El BotManager muestra en consola cada acción que realiza. Eso significa que se podría utilizar esta información para obtener estadísticas y tomar medidas de corrección de ser necesarias en el diseño del mundo virtual ya sea de aprendizaje, simulación, etc. Un ejemplo de esto sería el siguiente: si los usuarios muy frecuentemente necesitan que un NPC demuestre cómo realizar una práctica específica, porque no saben cómo realizarla, es posible que la práctica no sea muy intuitiva o la explicación de esta no esté muy clara.

Una evolución más futurista, pero posible de alcanzar con la tecnología actual, sería la integración del BotManager con dispositivos *hápticos*[1], de tal forma que un NPC podría tocar a un avatar y el usuario sentir el impacto a través de estos dispositivos, o escuchar mensajes que el NPC le dice a su avatar.

[1] dispositivos que proporcionan retroalimentación al usuario referente a los sentidos de vista, tacto y oído

Bibliografía

[Al-Gharaibeh and Jeffery, 2010] Al-Gharaibeh, J. and Jeffery, C. (2010). Portable non-player character tutors with quest activities. In *Virtual Reality Conference (VR), 2010 IEEE*, pages 253–254. IEEE.

[Antoniou and Van Harmelet, 2004] Antoniou, G. and Van Harmelet, F. (2004). A semantic web premier. *England: The MIT Press Cambridge*.

[Bartle, 2004] Bartle, R. A. (2004). *Designing virtual worlds*. New Riders.

[Brown and Cairns, 2004] Brown, E. and Cairns, P. (2004). A grounded investigation of game immersion. In *CHI '04 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '04, pages 1297–1300, New York, NY, USA. ACM.

[Criado and Tuset, 2014] Criado, M. Á. P. and Tuset, M. d. C. T. (2014). Mundos virtuales y avatares como nuevas formas educativas. *Historia y comunicación social*, 18:469–479.

[Dillenbourg et al., 2002] Dillenbourg, P., Schneider, D., and Synteta, P. (2002). Virtual Learning Environments. In Dimitracopoulou, A., editor, *3rd Hellenic Conference "Information & Communication Technologies in Education"*, pages 3–18, Rhodes, Greece. Kastaniotis Editions, Greece.

[Fernández-Avilés Pedraza, 2013] Fernández-Avilés Pedraza, D. (2013). Diseño de una práctica para un laboratorio virtual de biotecnología multilingüe y adaptable a alumnos de secundaria.

[González and Duque, 2008] González, H. M. and Duque, N. D. (2008). Modelo del estudiante para sistemas adaptativos de educación virtual student model for adaptive systems of virtual education. *Avances en Sistemas e Informática*, 5(1):199–206.

[Gros Salvat, 2004] Gros Salvat, B. (2004). La construcción del conocimiento en la red: límites y posibilidades.

[Gruber, 1993] Gruber, T. (1993). What is an ontology.

[Johansson et al., 2014] Johansson, M., Strååt, B., Warpefelt, H., and Verhagen, H. (2014). Analyzing the social dynamics of non-player characters. In *Frontiers in Gaming Simulation*, pages 173–187. Springer.

[Korolov, 2013] Korolov, M. (2013). U.s. army goes virtual with opensim. *Network World*, 30(5):24–27. Nombre - Army-US; Copyright - Copyright Network World Inc. Mar 25, 2013; Características del documento - Photographs; Última actualización - 2013-04-17; SubjectsTermNotLitGenreText - United States–US.

[Luzardo and Hernández, 2010] Luzardo, G. and Hernández, J. (2010). Inteligencia artificial en ambientes virtuales: Humanos virtuales autónomos (hva) como agentes virtuales inteligentes (3diva).

[Messinger et al., 2008] Messinger, P., Stroulia, E., and Lyons, K. (2008). Virtual worlds research: past, present & future. *Journal of Virtual Worlds Research*, 1(1):2–18.

[Moreno Sabido, 2002] Moreno Sabido, M. R. (2002). *Modelado del Estudiante para un Ambiente de Aprendizaje de Lecto/Escritura*. Universidad de las Américas Puebla.

[Papadopoulos et al., 2013] Papadopoulos, L., Pentzou, A.-E., Louloudiadis, K., and Tsiatsos, T.-K. (2013). Design and evaluation of a simulation for pediatric dentistry in virtual worlds. *Journal of medical Internet research*, 15(10).

[Párraga, 2011] Párraga, J. C. (2011). *Una propuesta de modelado del estudiante basada en ontologías y diagnóstico pedagógico-cognitivo no monótono*. PhD thesis, Universidad Politécnica de Madrid.

[Riofrío Luzcando, 2012] Riofrío Luzcando, D. (2012). *Diseño e Implementación de un Laboratorio Virtual de Biotecnología.* PhD thesis, Informatica.

[Thurau et al., 2006] Thurau, C., Hettenhausen, T., and Bauckhage, C. (2006). Classification of team behaviors in sports video games. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 1, pages 1188–1191.

Apéndices

Apendice 1

.1. Entorno de Desarrollo en C#

.1.1. Creación y Configuración de las Solución y Proyecto BotManager en C#

Dentro del entorno de desarrollo Monodevelop versión Xamarin se crea una nueva solución. Al crear la nueva solución también se un proyecto un proyecto con el mismo nombre, como se lo aprecia en la figura 1.

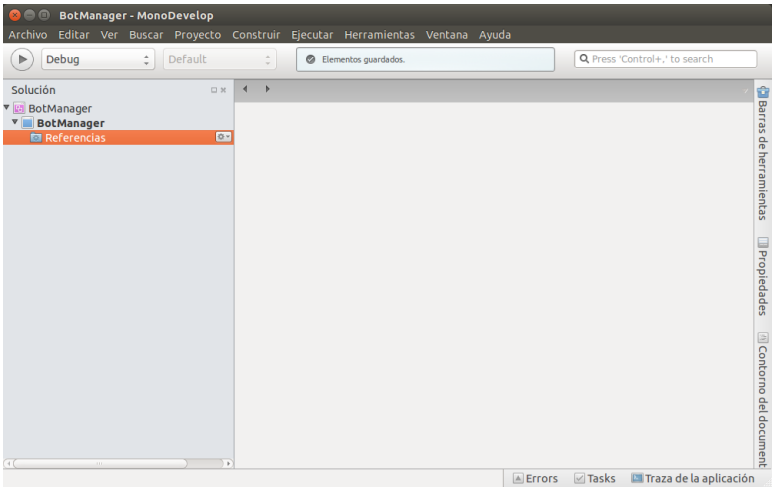

Figura 1: Creación de la solución BotManager en Monodevelop

El siguiente paso es añadir las librerías de cada componente para trabajar en el proyecto BotManager. Primeramente en el proyecto BotManager, crear la carpeta *resources* y dentro de esta la carpeta *libraries*.

.1.1.1. Referenciando OpenMetaverse en BotManager

En el proyecto BotManager agregar las siguientes referencias de librerías de OpenMetaverse, que se encuentran en de la ruta *$/libopenmetaverse-0.9.2/bin*:

- OpenMetaverse.Utilities.dll
- OpenMetaverse.Tests.dll
- OpenMetaverse.Rendering.Simple.dll
- OpenMetaverse.Rendering.Meshmerizer.dll
- OpenMetaverse.GUI.dll
- OpenMetaverse.dll
- OpenMetaverseTypes.dll
- OpenMetaverse.StructuredData.dll
- OpenMetaverse.Rendering.Linden.dll

.1.1.2. Referenciando a OpenSimulator en BotManager

Se deberán copiar dentro del proyecto BotManager en la carpeta *BotManager/bin/Debug*. Los siguientes archivos desde *$/OpenSimulator/bin*:

- libopenjpeg-dotnet-2-1.5.0-dotnet-1.dylib
- libopenjpeg-dotnet-2-1.5.0-dotnet-1-i686.so
- libopenjpeg-dotnet-2-1.5.0-dotnet-1-x86_64.so.

.1.1.3. Referenciando Jena en BotManager

Agregar las referencias de librerías de Jena, que se se encuentran en la ruta *$/LinkedDataJena/Dependencies*:

- LinkedDataTools.JenaDotNet.dll
- IKVM.OpenJDK.Core.dll

.1.1.4. Referenciando Librerías complementarias

Se deberán copiar dentro del proyecto BotManager en la carpeta *BotManager/bin/Debug*. Los siguientes archivos desde *$/OpenSimulator/bin*:

- log4net.dll
- XMLRPC.dll

.1.1.5. Referenciando a openmetaverse_data

Se deberán copiar dentro del proyecto BotManager en la carpeta *BotManager/bin/Debug*, la carpeta ubicada en *$/OpenSimulator/bin/openmetaverse_data*.

.1.1.6. Referenciando el Archivo de Propiedades Globales

Se debe copiar el archivo *resources.properties* en la carpeta *BotManager/bin/Debug*. El archivo *resources.properties* es un archivo de texto, que contiene los parámetros de configuración de la aplicación.